FACTORS OF PROGRAM COMPLEXITY AND THEIR EFFECTS ON PROGRAM COMPREHENSION

By

BANSHI DHAR CHAUDHARY

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JULY, 1979

FACTORS OF PROGRAM COMPLEXITY AND THEIR EFFECTS ON PROGRAM COMPREHENSION

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

By

BANSHI DHAR CHAUDHARY

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JULY, 1979

DEDICATED TO

MY PARENTS

EE-1979-D-DES-FAC

CERTIFICATE

This is to certify that the work entitled "Factors of Program Complexity and their Effects on Program comprehensio by Banshi Dhar Chaudhary has been carried out under my supervis and that this has not been submitted elsewhere for a degree.

H. V. Sahasrabuddhe

(Hari V. Sahasrabuddhe)
Professor
Computer Science Programme
Indian Institute of Technology
Kanpur 208016, INDIA

ACKNOWLEDGEMENTS

Words can not express the debt of gratitude I owe to my thesis supervisor Professor Hari V. Sahasrabuddhe. He has been a source of constant inspiration throughout the period of this work. His innovative ideas, able guidance and constructive criticism were invaluable towards the completion of this thesis.

I am thankful to Dr. Ramadhar Singh for his guidance and help in experiment design and analysis of data. His comments and criticism have greatly improved the manuscript.

I would like to thank Prof. V. Rajaraman and Dr. R.M.K. Sinha for their encouragements and abiding interest they showed in this work. I wish to thank my friends Dr. R.C. Sharma, Dr. S.N. Sinha, Mr. R.C. Gupta and Mr. B.H. Jajoo for their help. Their friendship had made my stay at Kanpur memorable.

My thanks are to my wife Baidehi and daughter Archana and Richa for their support. In particular, my wife has been as patient and as understanding as is humanly possible during years of my graduate study. This accomplishment is as much hers as mine.

The financial assistance of Government of India under Quality Improvement Program is gratefully acknowledged.

I am thankful to Dr. Jagdish Lal, Principal, M.N.R. Engineering College, Allahabad for sponsoring me under Q.I.P. for my doctoral degree.

I also wish to express my thanks to Sri K.N. Tewari for excellent typing of the manuscript.

BChaudhary
B.D. Chaudhary

Kanpur
July 1979

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SYNOPSIS

of the

Ph.D. Dissertation

on

FACTORS OF PROGRAM COMPLEXITY AND THEIR EFFECTS ON PROGRAM COMPREHENSION

by

Banshi Dhar Chaudhary
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

July 1979

A prime concern of the software industries has been development of programs which are easy to understand and maintain. Through a desire to reduce the perceived complexity of programs so as to make them easily understood, this has led to various studies directed towards identification of factors contributing to psychological complexity of programs and development of complexity-reducing methodologies.

Program understanding may be viewed as two stage process. In the first (interpretation) stage, the explicit facts about the program are abstracted for mental representation. In the second (learning) stage, various meta-inferential techniques are applied to search for rules (implicit facts) hidden under the gathered explicit facts by suggesting hypothesis for test and eventual confirmation into such rules. Several variants of the above model have been presented in literature as models of program understanding.

The psychological complexity of programs will be affected by all those factors which either aid or hinder the interpretation and learning of the program. The features of the program which suggest an analogy or establish an association with a problem domain are expected to facilitate learning by reducing the effort in search of hypothesis. These features of the program are contained in comments, variable names, data and control organizations, etc., and are collectively termed as 'meaningfulness' of the program.

The abstraction of facts about a program may involve mental expansion of program text in branch free sequences of statements or blocks followed by mental execution of these blocks. Control structure and data structure are expected to play an important role in formation of above blocks and their mental execution. They may also aid in 'meaningfulness' of the program. The limited capacity and volatile nature of human short-term memory make the size of the program an important factor of complexity.

This thesis establishes these four algorithm dependent features: size, control structure, data structure and computation structure of program as factors of program complexity and studies their effect on program understanding. The thesis also studies the effect of 'meaningfulness' of programs on their understanding.

The formal abstraction of structures of program objects or data are claimed to have important bearing on program understanding. Many new programming languages (CLU, ALPHARD, etc.) have been designed to incorporate the methodology of data abstraction. Some formal aspects of data abstraction have been studied in the thesis.

The major work of the thesis is outlined below along with results and conclusions.

A four factor judgemental experiment was conducted with experienced programmers as subjects to study the contribution of factors of size, control structure, data structure and computation structure to program complexity. These factors will be referred as 4-factors of program. Information integration theory was applied to identify the underlying rule of integration of various factors of program complexity. The result of the experiment was:

(1) Four factors of the program contributed independently to perceived program complexity.

A second experiment was conducted to investigate the effect of 'meaningfulness' of the expressions and the number of operands in the expressions on their evaluation. The 'meaningfulness' of the expressions were due to their <u>referential</u> meaning in some problem domain. The results of the experiment were:

(2) It was easier to evaluate meaningful expressions than the meaningless ones.

(3) The evaluation of expression was affected by number of operands in them.

The purpose of the third experiment was to study the effect of meaningfulness of the program and number of vectors, matrices, etc., referred to as numerosity of data structures, on program understanding. The value of meaningfulness was controlled by selecting variable names which were suggestive of their meaning and purpose and familiarity of the subjects with the problem. The results were:

(4) The 'meaningfulness' of the program facilitated their understanding. However, there was one interesting exception to this conclusion.

(5) The numerosity of data structures in the program did not affect program understanding significantly.

The above result (5) and the exception to conclusion (4) are against the intuitive expectation. The results of the further experiments indicated that these unexpected results were due to special features of the program used in the experiment. The uncontrolled feature dominated over the controlled factors.

Two measures of program comprehension were also compared in the experiment.

These above experiments indicated the overwhelming strength of influence of 'meaningfulness' factor. It was

decided to use programs which were as uniformly meaningful as possible in the experiments. Many programming standards of real life endeavour to achieve just this situation.

A 3x3 factorial experiment was conducted to study the effects of control and computation structures on program understanding. The results of the experiment were:

(6) Factor of computation structure (controlled by number of assignment statements and the number of operands in each) affected program understanding.

(7) The numerosity of control statements in the programs did not increase their control structure complexity. The regularity in the syntax and semantics of the control structure of the progam facilitated understanding.

The above three experiments were 1- and 2-factor experiments. The subjects of the experiment 2 and 3 were novice programmers. These experiments did not permit the study of interaction among the factors. The class-room environment and the strict time schedule of the course precluded use of large programs and inclusion of more factors in the experiment.

A four factor experiment was conducted with experienced programmers as subjects. The results of that experiment were:

(8) All the factors except program size and some of their interactions affected program understanding. The differences in the size of the programs were not large enough to affect program understanding.

In another experiment, the sixteen programs used in the previous experiment were subjectively rated for their complexity. The results, in general, confirmed the findings of the previous experiment.

The results of the experiments suggest that all the four algorithm dependent features of the program along with 'meaningfulness' of the program affect program understanding. Quantitative syntactic measures for complexity of the factors do not represent their contribution to psychological complexity of program.

The limited data structuring constructs of FORTRAN did not permit the experimentation with various aspects of factor of data structure complexity. As an alternative to experimentation, different specification techniques for data abstraction were studied theoretically and compared for their good and bad features.

(9) We have proved that Parnas' earliest (1972) specification technique is powerful enough to specify any computable function. Further, it has been proved that the completeness and consistency of the specification are not decidable.

Finally, some modifications are suggested in the above technique to facilitate the specification of any exception handling strategy and 'delayed effects'.

The thesis, in the end, also discusses the limitation of the present work and suggests direction for further research.

CHAPTER 1

INTRODUCTION

Program debugging, maintenance and modification are major activities of software industries. An observation of Elshoff, [Elshoff,76a,b] in a study of General Motor installations, that seventy-five percent of all programmers' time was spent on maintenance and modification of existing programs underscores the importance of above activities. An important prerequisite for debugging, maintenance and modification of programs is their comprehension. It is now widely felt that the complexity of programs should be significantly reduced to facilitate their understanding. This has led to various studies directed towards identification of factors contributing to complexity of programs and development of complexity-reducing methodologies. The complexity referred above is not the computational complexity of the program but its <u>psychological</u> complexity. These two complexities should be clearly distinguished. The computational complexity characterizes the order of time and storage requirements of the program. On the other hand, the psychological complexity refers to those characteristics of program which make human understanding of the program difficult. No direct relationship is expected between these two types of complexity.

Psychological complexity of programs is one of the concepts which one knows well by intuition but finds difficult to formalise. This difficulty is mainly due to absence of formal theory of program understanding. No general model has emerged either from empirical studies conducted to identify various factors of program complexity or from the research in the field of Artificial Intelligence. However, some common features have emerged from various models described in the literature.

In the model proposed by Mills, [Mills, 72], one major step in understanding a program text is the mental expansion of the text into branch free text sequences, corresponding to classes of execution using identical instruction sequences. A second major step is the mental execution of instructions along these branch free sequences. These two steps are often intermingled in mental visualisation of the process invoked by the text.

The model proposed by Bobrow and Brown, [Bobrow, 75 b], consists of two separate programs one to synthesize a model of the environment based on input data and other to analyze the information in the synthesized model in order to answer the questions about the environment. The synthesized model of the environment is the mental representation of the external world and its structure depends on the nature of the questions to be answered. Analyst retrieves knowledge

explicitly represented in the above model and uses various inferential techniques to infer facts not explicitly contained in the model.

Löfgren, [Löfgren, 77], makes distinction between an object and a description. The distinction is based on the process of learning. In the case of an object, learning proceeds by finding the regularity in its structure without transforming it to any other object of other domain. In the case of description, learning is preceded by transformation of the description into object of other domain. This distinction between an object and a description is only a way of looking at the things. The more regularities or structures are found, the more genuine is the learning and less complex is the object learned.

Shneiderman's syntactic/semantic model of programmer behaviour, [Shneiderman, 77b], hypothesizes that the programming experience and education build two kinds of knowledge structure. Syntactic knowledge is language dependent, acquired through rote memorization and must be frequently rehearsed to prevent forgetting. Semantic knowledge is language independent acquired through <u>meaningful learning</u> and heirarchically organized from low level constructs such as meaning of the assignment statement or an arithmetic expression, to intermediate program structures such as the sequence for clearing or sorting an array, to

higher level problem domain events such as the computation of a matrix inverse or a correlation coefficient.

Brooks, [Brooks, 77], has presented an explicit model for the coding process based on information processing theory of Newell and Simon, [Newell, 72]. The theory hypothesizes that three distinct sorts of behaviour, understanding, method finding and coding, are involved in performing a programming task. Understanding is characterized by acquisition of knowledge of the basic elements of the problem. These include the objects with which the problem is concerned, their properties and relations, initial and final state of the objects and the operations available for going from initial to final states. Extracting information from external sources and building of internal representation during which new information is incorporated are intermingled in the process of understanding.

It is evident from the above models that understanding is viewed as two stage process. We shall call the two stages as interpretation and learning stages. In the interpretation stage, explicit facts about the program are abstracted from the text of the program. This may involve mental expansion of the text into branch free instruction sequences and mental execution of these sequences [Mills, 72]. In the learning stage, one searches for rules (implicit facts) hidden in explicit facts by suggesting hypothesis

for test and eventual confirmation into such rules. A rule is considered more powerful the more explicit facts it concerns.

The psychological complexity of the program will be affected by all such factors which either aid or hinder the interpretation or learning of the program. Various factors such as comments, mnemonic names, paragraphing, different control constructs, have been studied for their effect on program understanding. Defining a complexity measure on the basis of such factors may prove difficult due to their numerosity.

We believe that the various factors affecting the program understanding may be synthesized in small number of higher level factors. The complexity measure of these higher level factors can be defined on the basis of their effect on program understanding.

The major contributions of this thesis are in identifying the various factors of program complexity and studying their effect on program understanding.

The features of the program which suggest an analogy or establish an association with a problem domain are expected to facilitate learning by reducing the effort in search of initial hypothesis to characterize the program structure. These features of the program are contained in comments, variable and subroutine names, data and control

organizations, etc. The comments help in establishing an association with problem domain objects and events. Similarly, selection of variable names for data and their organization are suggestive of the association between program and the problem domain. The control structure and execution profile of the program may also help in establishing this association. These features of the program are collectively termed as factor of 'meaningfulness' of the program.

The control structure and data structure of the program are also expected to play a major role both in interpretation and learning stages of understanding. More complex the control structure, more difficult it is to mentally expand the program text into branch free sequences of instructions and identify them. The mental execution of branch free sequences of instructions helps in abstracting the details of the program and in establishing a correspondence with the events of the problem domain. In the execution, the values are retrieved from the structure of the mental object. The structure of the mental object is expected to be dependent on the data structure of the program. Meaningful variable names and organization may aid in structuring the mental representation.

The other important factor, which will affect understanding, is size of the program. There may be some reservations in treating size as a separate factor. The

increase of control and assignment statements will increase the size as well as control and executional complexity of the programs. However, due to limited capacity and volatile nature of mental storage, size becomes an important factor after some threshold level. In a study by Thayer,[Thayer,75], it has been shown that program size is correlated with occurrences of actual software errors. This observation further underscores the importance of size.

These informal arguments suggest that the size, the control structure, data structure, and execution structure of a program will affect the program understanding and hence the program complexity. The above four factors are algorithm dependent. The factor of 'meaningfulness' will also have an important bearing on program comprehension.

This research establishes size, control structure, data structure and execution structure of programs as factors of program complexity through a judgemental experiment with experienced programmers. The above factors will be collectively referred as "4-factors of program complexity" in this thesis. The effect of 4-factors on program understanding has also been studied.

The data abstraction methodology is claimed to facilitate the understanding of data structures by formally abstracting their properties. Some formal aspects of data abstraction have been studied.

The next section includes a brief survey of the experiments on program comprehension and different complexity metrics. The final section describes the organization of the thesis.

1.1 BRIEF SURVEY

Weissman,[Weissman,73,74], in his experiment studied the effects of comments, paragraphing and mnemonic variable names on program understanding. The experiment was conducted with students in a computer science course. Several versions of two programs having different levels of factors of interest were used in the experiment. Three measures of program understanding: (i) how well the subjects hand simulated the program,(ii) how well they were able to fill in the blanks in a paragraph describing a program and (iii) the subjects own evaluation of how well he understood the program, were used in the experiment. The ranking of one's own understanding of the program was done on a scale of 0-9. The results of the experiment indicated the strong dependence of program understanding on these three factors. Comments and paragraphing facilitated understanding. Mnemonic variable names made both comprehension and modification easier than non-mnemonic names. However, the mnemonic variable names tended to slow down the subjects in hand simulation. A large number of errors were made in a combination where comments were present and mnemonic

variable names were absent. The interaction effect between paragraphing and comments was against the intuitive expectation. Paragraphing hindered understanding in presence of comments whereas it helped understanding in their absence.

Ben Shneiderman had investigated several programming issues such as program composition, comprehension, debugging and modification. In his early experiment [Shneiderman, 76], he compared the effect of Logical IF vs. arithmetic IF statement of FORTRAN on program understanding. The programs constructed using logical IF's proved easier to understand than the programs constructed using arithmetic IF's for novice programmers. However, there seemed to be no difference in the two syntactic forms for experienced programmers.

In another experiment he studied the program memorization task. Two short FORTRAN programs were used in the experiment. One was a proper executable program whereas the other contained valid statements in scrambled order. The results indicated that the program structure facilitated comprehension and memorization. As experience increased, the ability to memorize the proper program increased rapidly while ability to memorize the shuffled program showed minimal change.

The other results of his series of experiments were:

(i) Detailed flowcharts were found to be of no detectable

value in program composition, comprehension, debugging or modification[Shneiderman, 77a].

(ii) High level comments were significantly better than low level comments in aiding memorization; but in debugging, the presence and absence of comments seemed to be irrelevant [Shneiderman, 77b].

(iii) In difficult programs, mnemonic variable names made comprehension much easier than non-mnemonic names for experienced programmers. However, in simple programs it did not have any effect.

(iv) Modular programs were less difficult to comprehend than non-modular ones. The random modular programs were most difficult to comprehend among the three Shneiderman,78 .

(v) In debugging, the subjects performed better with indented unfamiliar programs than with non-indented programs [Shneiderman, 79].

Sheppard et al., [Sheppard,78], studied the effect of mnemonic variable names, program class and structure on programmers understanding of programs. Three general classes of programs used were: engineering, statistical, and non-numeric. Three levels of program structure defined were: structured, quasi-structured, and unstructured. The structured level adhered strictly to the tenets of structured programming [Dijkstra, 72]. Program flow proceeded

from top to bottom with one entry and one exit. Neither backward transfer of control nor arithmetic IF's were allowed. In quasi-structured level, use of backward GO TO statements and multiple exits were permitted. Awkward constructions resulting from rigorous application of rules of structured programming were eliminated. The unstructured level programs contained many GO TO statements. Backward transfer of control was not restricted and arithmetic IF's were allowed. The results of the experiment indicated that program class and program structure affected program understanding while no relationship was found for mnemonic variable names.

Some similar studies were conducted to study program construction. Sime, Green and Guest, [Sime, 73], compared the implementation of conditionals. Subjects were required to write five increasingly difficult programs. Subjects were divided in two groups. Both the groups used the two implementation of conditional alternatively to write programs. The group using implementation in a structured IF-THEN-ELSE manner had less difficulty constructing the programs than the group using implementation via tests and forward branch statements.

Brooks' work attempts to develop a theoretical frame work for the study of cognitive process involved in understanding, method-finding and coding process in program writing [Brooks, 77]. His experimental results had

demonstrated the sufficiency of the theory in explaining the observed programming behaviour in coding process.

A limitation common to most of the experiments surveyed above is that they are all single factor experiments. These single factor experiments do not permit the study of interaction among the factors. The various factors studied are not algorithm dependent.

In parallel with these works, several complexity metrics were developed for programs. The first theory was proposed by Halstead,[Halstead, 75], which states that programs have measurable characteristics analogous to physical laws. He defined the program size in terms of four basic measures: (i) number of distinct operators, (ii) number of distinct operands, (iii) total number of occurrences of operators, and (iv) total number of occurrences of operands in the program. The language level, programming effort and programming time are also defined on the basis of these four measures. The correlation between program size and mental effort required to generate the program is established on the basis of hypothesis that the program is generated by selecting items, such as operators, operands, etc., from language vocabulary by means of binary search.

Ronback,[Ronback, 75], associates a measure with control flow of the program to characterize its hierarchical

structuredness. The structuredness of the system is defined in terms of number of levels in the hierarchy to structure the program. An ideal structure which maximizes the number of levels would require only two components to make a subassembly at the next level. The absolute structuredness of a system can be compared to an ideal structure to get the ratio of structuredess. The number of absolute levels in the system is computed from the control flow graph of the program. In defining structuredness, the complexity of the proof of the correctness at any node of any level is also taken into account. The above measure has not been tested for its validity.

Amster et al.,[Amster, 76], used percent of the statements that affect control flow as a measure of complexity. However, this measure has a problem because complexity can be held constant as the size of the program increases.

McCabe, [McCabe, 76], has defined the complexity measure in relation to the decision structure of the program. His complexity metric, V(G), is the classical graph-theoretic cyclomatic number defined as: number of edges - number of nodes +number of connected regions. Sheppard et al., [Sheppard, 78], evaluated the relationship between comprehensibility and three program metrices (i) Halstead's measure of programming effort, (ii) McCabe's cyclomatic number of control flow graph, and (iii) the number of

statements in the program. It was found that both Halstead's and McCabe's measures were related to program understanding.

Woodward et al., [Woodward, 79], has forwarded "knots" as a measure of control flow complexity of programs. The knot is intersection of directional lines representing the flow of control. The knots can be defined mathematically as follows. If a jump from line a to line b is represented by ordered pair of integers (a,b), then jump (p,q) gives rise to a 'knot' with respect to jump (a,b) if either

(i) $\min(a,b) < \min(p,q) < \max(a,b)$ <u>and</u> $\max(p,q) > \max(a,b)$ or,

(ii) $\min(a,b) < \max(p,q) < \max(a,b)$ <u>and</u> $\min(p,q) < \min(a,b)$.

The count of number of "knots" in a program gives a measure of complexity.

In our opinion, the measures described above do not reflect the psychological complexity of programs. These measures reflect only numerosity aspect of control structure. The numerosity may be an important parameter in interpretation stage. But in the learning stage the regularity or pattern of control structure is more dominant. The regularity may exist at statement level, subroutine level or even at problem domain level. None of these metrices captures the difficulty in learning stage.

Further, the units used in defining the measures, such as operators, operands, basic paths, etc., do not seem to represent the cognitive units used by human beings for

program understanding. It seems more probable that people use higher level structures as cognitive units in understanding. However, we feel that these measures are well suited for program construction tasks because the basic units used in the task are same as used for the measures.

Other limitations of the control complexity measure are that they do not take into account the level of nesting and the effect of data values on the control flow. Association of data values with control paths seems to be an important part in program comprehension.

1.2 ORGANISATION OF THE THESIS

Chapter 2 of this thesis describes a four factor judgemental experiment with experienced programmers. The experiment was conducted to find whether the factor of size, control structure, data structure and computation structure contribute to overall program complexity. Information integration theory, [Anderson, 72], was applied to identify the underlying rule of integration of various factors for overall program complexity. The result indicated that 4-factors contribute independently to perceived program complexity.

Three experiments with novice programmers are described in Chapter 3. The first experiment studies the effects of meaningfulness of the expressions and the

number of operands in the expression on their evaluation. It is observed that the evaluation of meaningful expression is easier than meaningless ones. Further, the expression evaluation is affected by number of operands.

In the second experiment, the effect of factor of 'meaningfulness' and data structure complexity on program understanding were studied. The meaningfulness of the programs was controlled by selection of variable names which were suggestive of their meaning and purpose and problems of varying degrees of familiarity for the subjects. Results suggest that it was easier to comprehend the meaningful program. However, there was an exception to this conclusion. The numerosity of data structures did not show any significant effect on program understanding. This conclusion contradicted the intuitive expectation. However, the experiments of Chapter 4 explained the reasons for these unexpected results. Two measures of program comprehension were also compared.

The last experiment of the chapter studied the effect of control structure and execution structure complexities on program understanding. The programs used in the experiment were meaningful ones. The execution structure complexity affected the program understanding.

The experiments of Chapter 3 were 1- and 2-factor experiments and the subjects were novice programmers.

Interactions between factors could not be studied in these experiments. The class-room environment and strict time schedule for the course forced the use of small programs in the experiment and constrained us from designing multi-factor experiments.

Chapter 4 describes a 4-factor experiment with experienced programmers to study the effect of main factors and their interactions on program understanding. Another experiment was conducted to test the effectiveness of the subjective rating of program complexity.

The results of the experiment emphasized the importance of data structure in program understanding. The data abstraction methodology is claimed to facilitate understanding of data structures by abstracting their relevant properties.

In Chapter 5, a brief survey of various data abstraction techniques is given. It is proved that Parnas' early (1972) specification technique is powerful enough to specify any computable function. Further, the completeness and consistency of the specification are proved to be undecidable. Some modifications are suggested in Parnas' technique to facilitate the specification of exception handling strategy and 'delayed' effects.

The last chapter, Chapter 6, summarises conclusions from the major works of earlier chapters, discusses the limitations of the experiments and suggests possible directions of the future research.

CHAPTER 2

PERCEPTION OF PROGRAM COMPLEXITY

Informal but intuitive arguments were advanced in the previous chapter which identified size, control structure, data structure and execution structure complexities as contributing factors of overall program complexity. In this chapter an attempt is being made to empirically verify their influence.

The main purpose of the experiment described here was to answer the following questions: (a) Do factors of size, control structure, data structure and execution structure of the program contribute to program complexity? and (b) How do they contribute to program complexity? Answers to these two questions would help design other experiments on program understanding.

The organisation of this chapter is as follows. First section contains a brief introduction to information integration theory [Anderson, 72], the second section describes experimental method and design, and the final section includes analysis of data and discussion of the results.

2.1 INFORMATION INTEGRATION THEORY

Information integration theory is a unified, general theory of human judgement. Information refers to details about one's surroundings either in coded or uncoded form

on the basis of which judgements are made. Integration refers to the process whereby information about several coacting stimuli are combined to produce an overall response or judgement. The underlying idea of the theory can be described with the help of the diagram in Figure 2.1, reproduced from [Anderson, 77].

In Figure 2.1, S_1, S_2 and S_3 represent physical stimuli or external information available for judgement. These stimuli activate various sense organs and get converted into psychological stimuli, s_1, s_2, s_3. In the case of simple sensory stimuli, the conversion process is known as psychophysical law. In general, however, it is known as 'valuation function', V. The psychological values s_1, s_2, s_3 are combined by the individual in making judgements. This co-ordination process is known as 'integration function', I, and the judgement or response is known as 'covert response', r. This covert response is transformed by response function, M, into 'overt response', R.

The integration function, I, is a psychological law which relates the internal stimuli to the internal response. The response function, M, relates this covert response to the overt measurement scale imposed by the investigator. If M is a linear function, then R is said to be on interval scale.

In integration theory, simple algebraic models are used to describe the integration process, I. These simple

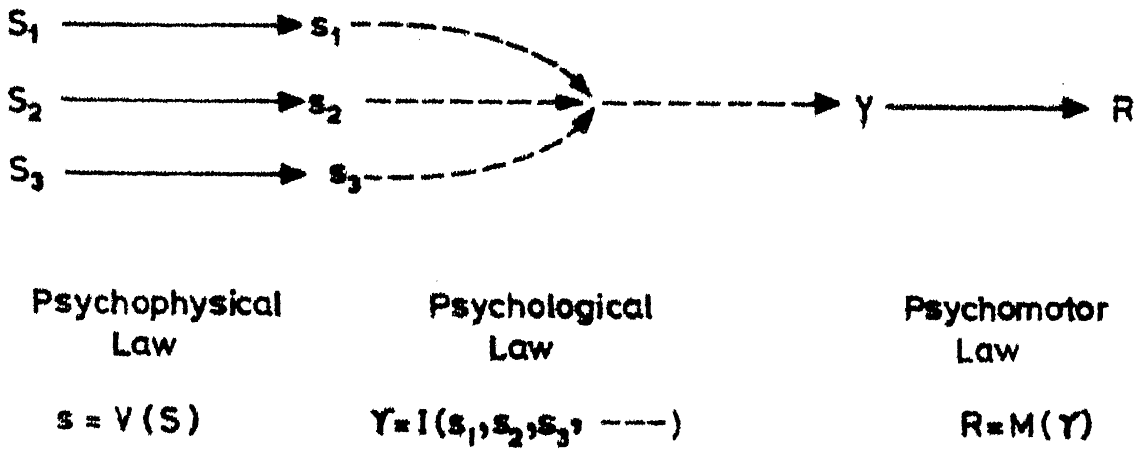


FIG. 2.1 FUNCTIONAL MEASUREMENT DIAGRAM

algebraic models of perception and judgement have given a detailed quantitative account of fairly complex cognitive activities.

Most of the models fall in one of the two main classes. One class includes adding, subtracting and averaging models; the other includes multiplying and dividing models. Adding and subtracting are formally similar. But they may be psychologically different. Adding and averaging are different both mathematically and psychologically. Under certain situations, however, they make identical prediction and have a very simple analysis. Similar is the case with multiplying and dividing models. Mathematically they are similar but psychologically they are different. Dividing models have been used in comparative judgements, while multiplying models have been employed in work on utility theory [Shanteau, 72].

In algebraic models, two stimulus parameters -- scale value (s) and weight (w), have special importance. The s is considered as the location of stimulus along the dimension of judgement. The w is viewed as the relative importance of different pieces of information coming into judgemental task. Both weight and value are dependent on the dimension of the judgement. Each task sets in a valuation operation by virtue of which the stimulus parameters are defined.

Since the averaging model of integration process has accounted for many facts about human judgements and has provided a general basis for estimating weight parameters, a brief discussion of the model is given below. In the averaging model, the response, R, to a set of stimuli can be written as

$$R = \sum_{i=o}^{N} \underline{w}_i \, \underline{s}_i / \sum_{i=o}^{N} \underline{w}_i \qquad (2.1)$$

The averaging model is a special case of linear model with the constraint that the relative or effective weight, $\underline{w}_i / \sum_{i=o}^{N} \underline{w}_i$, sum to unity. The development of integration theory has shown that it is necessary to allow for an internal state variable or general impression as well. This is treated as another stimulus, S_o, with value and weight, $\underline{s}_o$ and $\underline{w}_o$, respectively. In many cases, S_o represents the integrated result of the past experience and the current state in an ongoing task. However, S_o may also represent disposition as well as motivational variables. There are different cases of the averaging model, but the simplest is equal weight averaging. It is like a linear model and can be handled in a simple way.

Suppose that two stimulus variables, program size and its control structure, for example, are thought to combine in an additive or linear way to determine the overall complexity of the program. The two variables are

combined in an ordinary Row x Column factorial design. Several levels of one stimulus variable constitute the rows of the design and several levels of the other stimulus variable constitute the columns of the design. Each cell of the design thus corresponds to a pair of stimuli. This factorial design can be extended for any number of stimuli. The <u>equal weight</u> condition requires that all stimuli within each factor of the design have the same weight. For a two factor design, the row stimuli would all have a common weight, $\underline{w}_R$, and the column stimuli would all have a common weight, $\underline{w}_c$. When this condition is satisfied, then the averaging model reduces to

$$R_{ij} = C_o + \frac{\underline{w}_o\, \underline{s}_o + \underline{w}_R\, \underline{s}_{Ri} + \underline{w}_c\, \underline{s}_{cj}}{\underline{w}_o + \underline{w}_R + \underline{w}_c}$$

$$= \underline{C}'_o + \underline{w}'_R\, \underline{s}_{Ri} + \underline{w}'_c\, \underline{s}_{cj} \qquad (2.2)$$

where R_{ij} is the response due to combination of ith row and jth column stimuli C'_o is a constant which allows for an arbitrary zero on the response scale. $\underline{w}'_R$ and $\underline{w}'_c$ are relative weights of the row and column stimuli, respectively. Equation (2.2) ignores the error variability. The above equation implies that data should exhibit a simple pattern of parallel lines [Anderson, 74b].

The validation of integration model is straightforward. It requires a test of goodness of fit. If the integration

function is validated empirically then this function itself can be used as the base and frame for estimating subjective value of the stimuli. This idea is known as "functional measurement". This approach is used to answer the two questions posed at the outset.

There is a direct relationship between equation(2.2) and the structural model used in analysis of variance [Winer, 71]. The graphical prediction of parallelism is equivalent to a zero Row x Column interaction. If the parallelism is observed, that supports the linear model of equation (2.2). Further, it indicates that the response measure is on interval scale. That follows because unequal intervals would produce nonparallelism even with a correct model. Finally, it can also be shown that the row means of the design are interval scale estimates of subjective value of the row stimuli and similarly for column stimuli.

Similar estimates can be made from multiplying integration rule. Functional measurement for the multiplying model is the same as for the linear model. Multiplying rule may be hypothesized as

$$R_{ij} = C_o + \underline{s}_{Ri}\ \underline{s}_{cj} \qquad (2.3)$$

If $\underline{s}_{cj}$ were known, then each row of the data would plot as a straight line with a slope $\underline{s}_{Ri}$. Or if the $\underline{s}_{Ri}$ were known then each column of the data would plot as a straight line

with slope s_{cj}. In either case, the data would exhibit the form of linear fan. If the model is empirically supported, then the column means of the design are interval scale estimate of s_{cj}. In practice, therefore, the observed column means may be used as provisional values of s_{cj} to test the linear fan prediction.

2.2 EXPERIMENT DESIGN

In this experiment, subjects received information about the size, control structure, data structure and execution structure complexities of several hypothetical programs and gave their judgement about the overall complexity of each program. From the judgemental data, an integration function was hypothesized and empirically verified. This integration function in turn helped to assess the contribution of factors to program complexity and their interaction.

Some salient features of the experimental methodology are mentioned here. The response scale chosen was a 31-point graphic scale to facilitate fine discrimination among the programs. A small number of steps in the scale would have been susceptible to the end effects, whereas large number of steps would have encouraged lumping. It was expected that a 31-point scale would give an optimal response. The other advantages of using graphic scale are that it avoids the tendency for 'residual number preferences' and eliminates memory effects [Anderson, 74a].

To remove end effects and consequent nonlinearity in the response scale, stimulus end anchors were used. These end anchors were stimuli more extreme in either direction than the regular experimental ones. The end anchors were intended to enable the subjects to use entire response scale and to obtain regular experimental data from the interior of the scale.

As range of the stimuli and the response scale were arbitrary, some experience was needed to develop a frame of reference for the task. Preliminary practice and end anchors were employed to set the subjects frame of reference as is usually done in functional measurement.

To minimize the effect of the order of presentation of stimuli, the stimuli were presented in different shuffled orders to different subjects. The order and carry-over effects were further minimized by presenting the stimuli in two different orders. Some subjects received the description of the program in one order. Other received the program description written in the reverse of that order.

2.2.1 Methodology:

The details of the experiment and its design are presented below.

A. Stimuli:

Stimuli in the experiment were the information about the program size, its control structure, data structure and

execution structure complexities as described below under the design section. These four information constituted the four factors of the experiment. Each of the above four factors had three levels. In the case of program size, the levels were small, moderate and big. For control structure and data structure complexities, the levels were simple, moderately complex, and very complex. The last factor, execution structure complexity, had low, moderate, and high as the three levels. Since there were four factors, each having three levels, a total number of 81(3^4 = 81) stimulus combination were generated. These program description will be referred as four factor design. Each combination of the stimuli was typed on 4" x 6" index card. The different factors were listed in vertical order on the card.

There were 54 two factor program description also. These program descriptions contained information about only two of the four factors. Since these stimuli set had information about only two factors, it will be referred as two factor design. There were thus six types two factor program descriptions.

To control the probable effect of presentation of factors the above mentioned 81 four factor and 54 two factor program description were typed in reversed order also. Two typical stimulus card from each set are shown below.

Original Order

Program Size	=	Small
Control Structure	=	Moderately Complex
Data Structure	=	Simple
Executional Complex.	=	Moderate

Reversed Order

Executional Complex	=	Moderate
Data Structure	=	Simple
Control Structure	=	Moderately complex
Program Size	=	Small.

Along with these 135, (81+54), program descriptions, two end anchors were also presented. Practice set of stimuli contained two end anchors and eight other stimulus combination selected randomly from the 81. These end anchors and practice examples are given in Appendix A.

B. Subjects:

There were 12 subjects. Nine of them have been working as System Programmers at Computer Center of Indian Institute of Technology, Kanpur. Their experience in programming ranged from 2 to 10 years. They had a thorough experience with FORTRAN and Assembly language and good familiarity with COBOL, Autocoder, etc. Two of the subjects were senior graduate students in the Computer Science Program, working for their doctoral degree. They had approximately

3 years of experience in FORTRAN programming. They also knew COBOL and Assembly language. One subject was a senior faculty member in the Computer Science Program. He has been teaching programming courses at undergraduate and graduate levels for approximately six years. All these subjects had experience of repeatedly trying to understand others programs to help them in debugging and maintenance. This experience was gained by working as consultants, programming laboratory incharge or instructor. Their long experience of understanding and maintaining others programs made them a good choice as subjects for this experiment.

C. Procedure:

The experiment was conducted one subject at a time in a small room. On arrival, the experimenter explained verbally the purpose of the experiment. The subjects also received a typed sheet of instruction which described the nature of the experimental task and his role as subject. The instructions are reproduced in Appendix B. The subject was urged to understand the task clearly and to seek clarification in case he had any doubt. Each subject worked with ten practice examples. The subject was instructed to read the information about the four factors of hypothetical program, to form an opinion about its overall ecomplexity and to rate its complexity on a 31-point scale. After completion of practice examples, the subject received 137, (81+54+2) cards

containing the description of hypothetical programs. Cards were shuffled thoroughly. The subject read each card and rated its complexity. After rating all cards, a break of 20 minutes was given. The entire procedure was repeated (except the practice example) by the subject after the break. As pointed out earlier, there were two sets of cards containing the stimulus combination in the normal and reversed order. Half of the subjects worked on one set of the cards; the other half worked on the second set of the cards.

D. Design:

The experiment had two sets of stimulus design. One set had 3x3x3x3, program size x control structure x data structure x execution structure complexities, factorial design. This design we have referred as 4-factor design. Another set of design had six 3x3, program size x control structure, program size x data structure, program size x execution structure, control structure x data structure, control structure x execution structure, data structure x execution structure complexities, factorial design. This design has been referred as 2-factor design.

2.3 RESULTS AND DISCUSSION

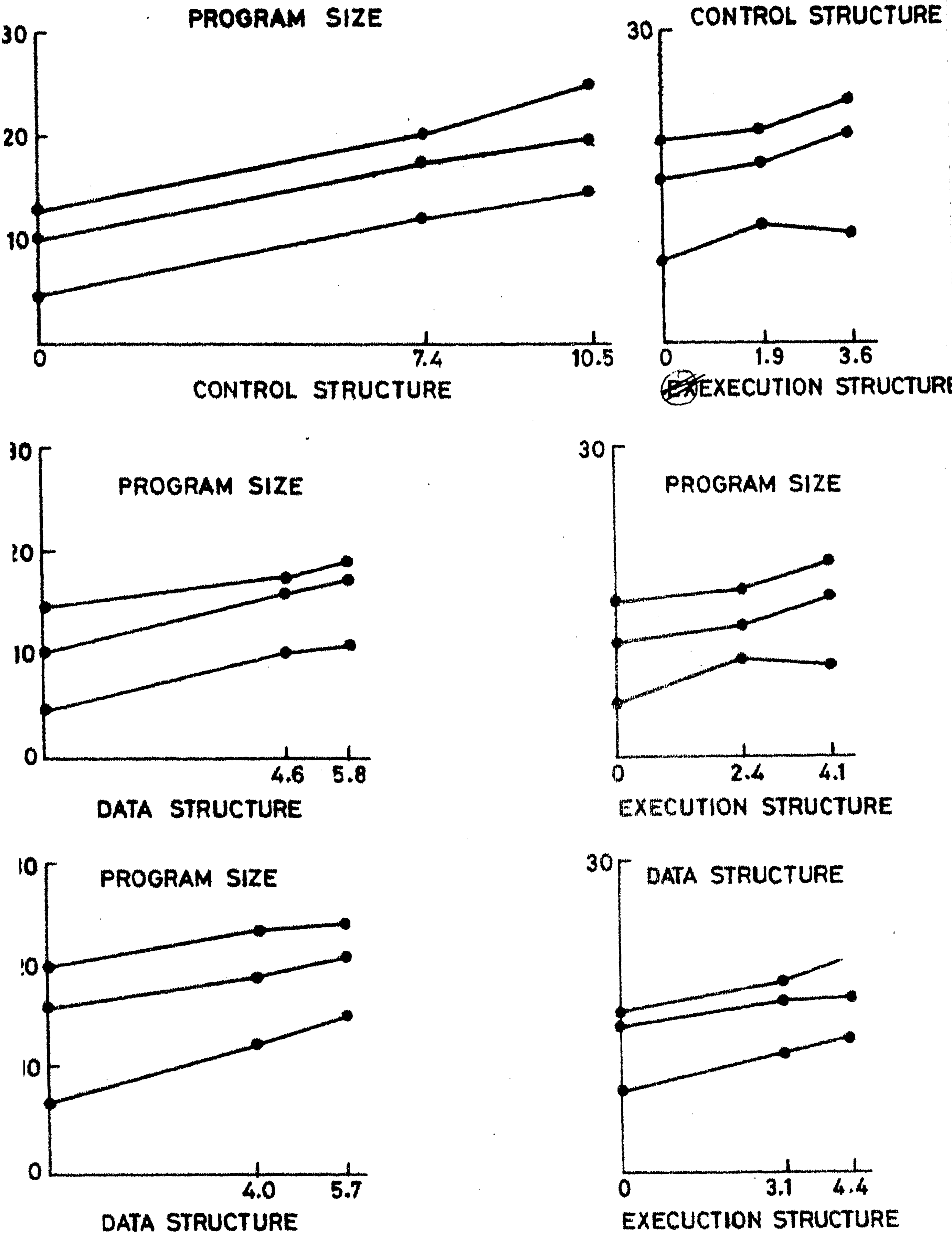
A. Graphical Analysis:

A simple and useful test of the integration model can be made by plotting the raw data. The mean program complexity was therefore plotted as function of program

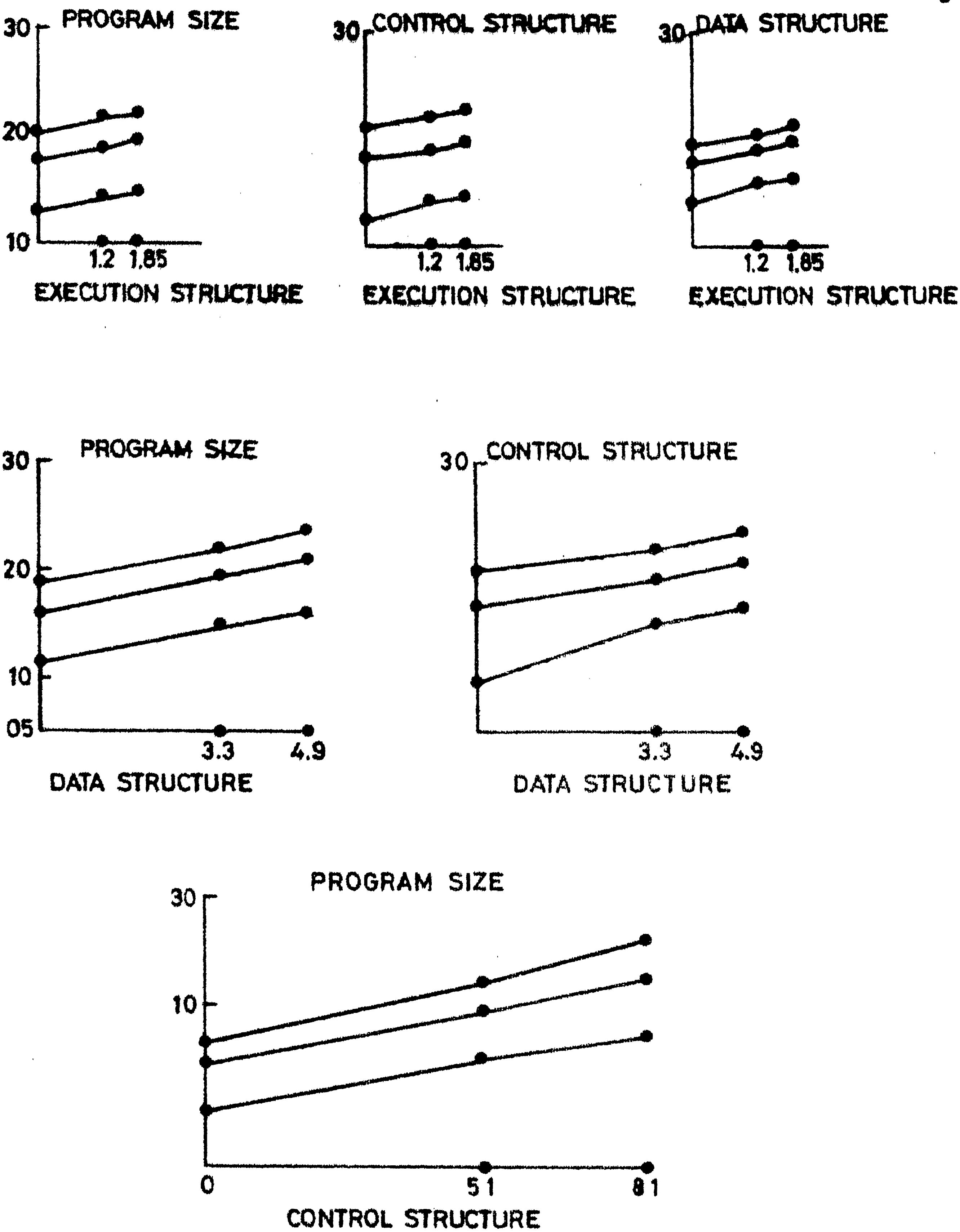
size x control structure, program size x data structure, program size x execution structure, control structure x data structure, control structure x execution structure, data structure x execution structure complexities. In Figures 2.2 and 2.3 the column levels are spaced according to their functional values [Anderson, 76]. These six sets of 2-way graphs of Figures 2.2 and 2.3 are from 2- and 4-factor design, respectively.

In general, the curves of Figure 2.3 are smoother than those of Figure 2.2. This is not surprising. The means used in Figure 2.3 were obtained by pooling a larger number of observations than those used in Figure 2.2.

The six sets of curves in Figure 2.2 do not exhibit a common trend. None of the curves show clear parallelism. It is therefore difficult to hypothesize any one of the integration models by just looking at the nature of the curves. If we hypothesize an additive integration rule, the deviations from parallelism have to be treated as minor non-linearity in response usage or idiosyncratic judgements of a few stimulus combinations. In fact, the deviation from parallelism in each set appears to be attributable to only one point of the set. On the other hand, if we decide to choose the multiplying model then these deviations from parallelism have to be treated as genuine. To arrive at any convincing interpretation, Figure 2.3 should be examined.



Y-AXIS REPRESENTS MEAN PROGRAM COMPLEXITY FOR ALL CURVES



Y-AXIS REPRESENTS MEAN PROGRAM COMPLEXITY FOR ALL CURVES

FIG.2.3 GRAPHS FROM 4-FACTOR DESIGN

The four sets of curves, program size x execution structure, control structure x execution structure, data structure x execution structure complexities, of Figure 2.3 clearly exhibit parallelism. The curves of program size x control structure have a diverging trend whereas the curves of control structure x data structure have a converging trend. In these two curves, also, the nonparallelism centers around one point only. Over the two designs, only curves of program size x control structure show a consistent pattern. Again it is difficult to choose between an additive and multiplying model from these graphic analysis.

B. Statistical Analysis:

To supplement the qualitative evidence from the graphical analysis, an overall analysis of variance was performed on the data of 4-factor design. In this 12x2x3x3x3x3 analysis of variance, the factors were subjects, replication, program size, control structure, data structure and execution structure complexities. For 2-factor design, six separate 12x2x3x3 analysis of variance were performed. In these analysis, the factors were subjects, replication and two of the four factors of 4-factor design. As each subject rated all the stimuli, interaction of the subject with a source of variance was used as error term in the construction of F ratios [Winer, 71, p.202].

F ratios for all 2-way interaction for both the designs are presented in Table 2.1. The interaction terms were significant in program size x execution structure, $F(4,44) = 2.67$, $p < 0.05$, control structure x data structure, $F(4,44) = 2.69$, $p < 0.05$, and control structure x execution structure, $F(4, 44) = 3.84$, $p < 0.05$, in 2-factor designs. In 4-factor design, the significant interactions were program size x control structure, $F(4, 44) = 6.36$, $p < 0.05$, and the control structure x data structure, $F(4, 44) = 4.28$, $p < 0.05$. The control structure x data structure interaction seems to be reliable. It is present in both 2- and 4-factor designs. According to an additive model, the interaction terms in the analysis of variance should all be nonsignificant. The two significant interactions mentioned above thus cast doubt on the applicability of an additive model.

To have a more powerful argument either for additive type model or multiplying model, it is appropriate to test the goodness of the fit of these two models to the data. The mean absolute deviations between the predicted and observed means was used as criterion for comparing the two models. The means from both 2- and 4-factor designs were prepared in a Row x Column table for all the six sets of 2-way interactions. The entry of each cell was observed mean, OV. To obtain the theoretical value, TO, for each cell, in the case of additive model, the marginal means for each

TABLE 2.1

F RATIOS FOR 2-WAY INTERACTIONS

S. No.	Interacting Factors	2-Factor Design	4-Factor Design
1	Program size x Control structure	1.51	6.36*
2	Program size x Data structure	1.80	0.22
3	Program size x Execution structure	2.67*	0.44
4	Control structure x Data structure	2.69*	4.28*
5	Control structure x Execution structure	3.84*	2.38
6	Data structure x Execution structure	1.35	1.31

* indicates statistically significant.

row and column and the grand mean were computed. Let $\bar{R}_{i.}$, $\bar{C}_{.j}$ and $\bar{G}_{..}$ denote ith row marginal mean, jth column marginal mean and grand mean. For any cell, ij, the theoretical value, TO_{ij}, is

$$TO_{ij} = \bar{R}_{i.} + \bar{C}_{.j} - \bar{G}_{..} \tag{2.4}$$

The mean absolute deviation was computed as follows:

$$E = \sum_{i} \sum_{j} |(OV - TO)| / n \tag{2.5}$$

In the present case n is equal to 9.

In the case of multiplying model, TO for each cell was estimated according to equation given below:

$$TO_{ij} = \bar{R}_{i.} \times \bar{C}_{.j} / \bar{G}_{..} \tag{2.6}$$

and the mean absolute deviation was again calculated according to Equation (2.5).

The mean absolute deviation from both the models are shown in Table 2.2. From entries in the table, it is evident that mean absolute deviations are less under additive model as compared to under multiplying one. This trend is true in all but one case. In other words, the additive model gave a better fit than the multiplying one.

To prove the superiority of additive over multiplying model, the absolute mean deviation from the predicted values of the two models were compared. The sixth column of Table 2.2 lists the *t* ratios for the difference between the two models.

TABLE 2.2

MEAN ABSOLUTE DEVIATIONS

Type of Design	S.No.	Row Factor x Column Factor	Additive Rule	Multiplying Rule	$\underline{t}$ Ratio
2-Factor	1	Program size x Control structure	0.463	0.784	1.59
	2	Program size x Data structure	0.495	0.85	1.94*
	3	Program size x Execution structure	0.604	0.704	0.2
	4	Control structure x Data structure	0.653	1.10	3.05*
	5	Control structure x Execution structure	0.610	0.6	0.17
	6	Data structure x Execution structure	0.458	0.603	1.36
4-Factor	1	Program size x Control structure	0.350	0.266	0.737
	2	Program size x Data structure	0.066	0.246	2.36*
	3	Program size x Execution structure	0.08	0.12	1.4
	4	Control structure x Data structure	0.603	0.910	3.85*
	5	Control structure x Execution structure	0.146	0.248	3.29*
	6	Data structure x Execution structure	0.126	0.187	2.904*

$t_{.95}(8) = 1.86$ (one tailed)

* indicates statistically significant.

It can be seen that the additive model appears to be superior to multiplying model in two cases of 2-factor design and in four cases of 4-factor design. From the results at hand, it seems reasonable to claim that an additive model gives a better account of the data than does the alternative multiplying model.

C. Single Subject Analysis:

To supplement the results of the previous sections and explain the minor discrepancy from parallelism in the main analysis mentioned earlier, separate analysis was made at the level of each subject. Individual analysis, it should be noted, are important to check that the group averages are not hiding alternative integration strategies by different subjects [Singh, 79], [Shanteau, 69]. Table 2.3 presents F ratios for all 2-, 3- and 4-way interactions for each subject in the main 4-factor design.

Inspection of Table 2.3 fails to reveal any meaningful pattern. It shows that significant interactions are only a few in number and that they are scattered. The 3- and 4-way interations are statistically significant in case of only one or two subjects. These subjects also disclose inconsistent patterns in the significant interactions. It appears reasonable, therefore, to accept our earlier interpretation of additive rule for the judgement of program complexity.

TABLE 2.3

F RATIOS FROM SINGLE SUBJECT ANALYSIS

Serial Number	Prog.sizex Cont.struct.	Prog.size x Data struct.	Prog. size x Exec. struct.	Cont. struct.x Data struct.	Cont.struct.x Exec.struct.	Data struct.x Exec.struct.	Prog. size x Cont.struct.x Data struct.	Prog. size x Cont.struct.x Exec.struct.	Prog. size x Data struct.x Exec.struct.	Cont.struct.x Data struct.x Exec.struct.	Prog.size x Cont.struct.x Data struct.x Exec.struct.
01	2.9*	0.33	1.0	17.6*	1.9	0.61	2.4*	0.4	0.47	0.89	0.84
02	1.10	0.66	0.51	3.71*	0.38	0.49	0.26	0.72	0.91	1.02	0.65
03	3.61*	2.51*	3.60*	0.81	0.91	0.36	3.73*	2.39*	2.53*	0.48	1.29
04	0.56	1.24	1.77	7.37*	0.89	2.86*	1.26	0.65	3.79*	1.66	0.96
05	2.13	5.76*	0.74	1.56	0.21	1.31	0.38	0.73	0.71	1.21	0.54
06	3.15*	2.81*	0.71	2.38	1.04	0.11	4.01*	0.51	1.54	2.34*	0.77
07	3.49*	1.37	1.30	0.32	0.72	2.67*	1.59	0.56	0.97	1.26	1.56
08	2.57*	1.00	0.81	9.09*	3.57*	1.94	0.17	0.35	1.38	1.08	2.37*
09	0.49	0.49	0.35	1.62	1.16	0.73	0.70	1.37	0.35	0.57	0.99
10	11.48*	0.68	0.39	0.40	0.90	0.19	1.05	0.78	1.30	0.84	0.93
11	5.75*	2.42	2.35	4.60*	3.27*	1.00	1.54	3.54*	1.18	1.53	1.14
12	0.78	1.14	0.38	1.40	0.26	0.17	1.46	0.83	0.31	0.33	0.58

* indicates statistically significant.

D. Supplementary Analysis:

Although the main purpose of the experiment was different, some of the results of this experiment have implications for information integration theory.

If all the stimuli have the same weight, averaging and adding models make identical prediction. A simple adding rule implies that every addition of information pertaining to program complexity should increase the overall program complexity. In contrast, an averaging model predicts a decrease in the overall program complexity when two pieces of information about the program are of unequal value. To plot all the six sets of 2-way curves, mean program complexity were obtained from 2- and 4-factor designs. In the case of 2-factor design, each mean is based on only two factors. But in the 4-factor design, each mean is based on the two factors as well as the average of the two other factors of the design. Since the mean program complexity in the case of 4-factor design is obtained by pooling over more number of factors compared to 2-facto design, the former should be larger than the latter in case adding rule is operative. But in averaging operation, the former should be lower than the latter.

To diagonise the real model, it is only necessary to plot the curves from 2-and 4-factor designs. If averaging rule is operative, then at least one of the curve of 2-factor design should cross over at least one of the curves of 4-factor design. This crossover is indicative of averaging rule[Anderson, 65].

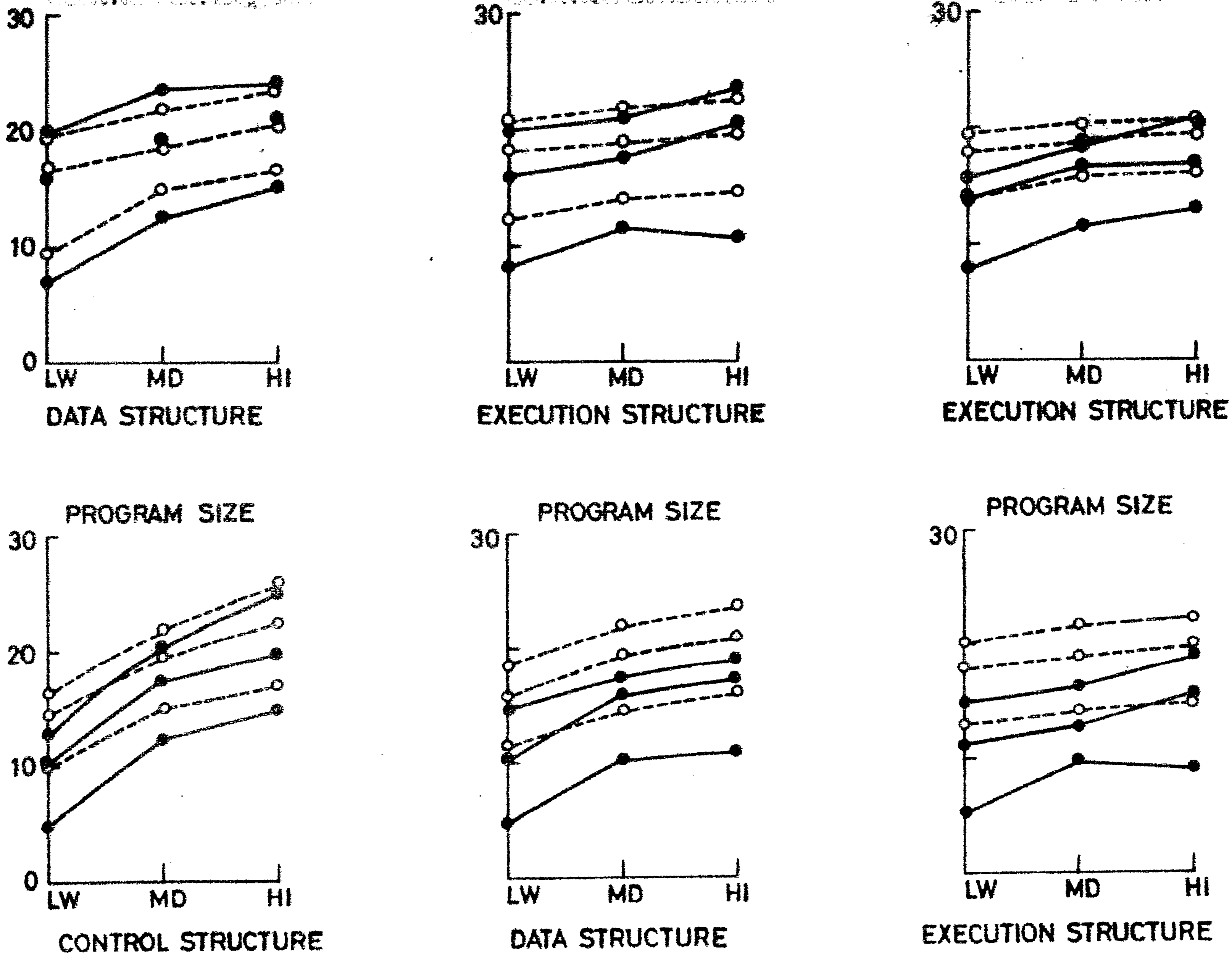
Six sets of 2-way curves for mean program complexity were plotted in Figure 2.4. The column levels are equally spaced. The dotted curves represent data from 4-factor design, whereas the solid curves represent from 2-factor design.

From the curves, it is evident that the rule of integration is averaging type. In all sets of the curves, solid lines cross-over the dotted lines. The result thus confirms the averaging rule and infirm the alternative adding rule.

This success of the averaging model is not surprising in view of the mass of evidence that supports the averaging hypothesis in many different domains [Anderson, 65], [Dalal, 78], [Singh, 75,79]. It is, however, interesting to note that the present success extends the generality of the averaging model to a completely novel judgemental task and to a different subject population. Accordingly, it can be said that averaging rule is perhaps a general way of co-ordinating information.

E. Conclusions:

From graphical analysis, analysis of variance, and from test of goodness of fit, we conclude that additive type model explains most of the experimental data. Selection of additive integration model accomplishes two goals simultaneously:



Y-AXIS REPRESENTS MEAN PROGAM COMPLEXITY FOR ALL CURVES

FIG.2.4 ADDING Vs. AVERAGING TEST

(i) It indicates that response is on interval scale and
(ii) It yields interval scales of the stimulus variables.

The question posed in the beginning can be answered very easily on the basis of this validated additive integration rule. It is easy to prove that marginal means of the data table are the stimulus values on validated interval scale. The marginal means along with F ratios of the four factors of the experiment are presented in Table 2.4.

TABLE 2.4 : MARGINAL MEANS OF FACTORS

S.No.	Factors	Level 1	Level 2	Level 3	F(2,22) ratios
1	Program size	14.05	18.71	21.33	38.75*
2	Control structure	13.65	18.73	21.71	29.58*
3	Data structure	15.32	18.60	20.18	45.13*
4	Execution structure	17.03	18.22	18.85	26.60*

* indicates statistically significant.

As all the four F ratios were highly significant, it can be said that the four factors of size, control structure, data structure, and execution structure complexities contribute to overall program complexity in the opinion of the programmers. This result suggests that the program complexity is dependent upon all these factors and that each factor makes an independent contribution to program complexity.

If we look at the marginal means of each factor, we find that the first and third values differ markedly across the four factors. This implies that the different factors were felt to vary in importance. To prove it clearly, ratios of sum of squares for each factor were taken with the total variance [Moll, 77]. These ratios for program size, control structure, data structure and execution structure complexities turned out to be 0.193, 0.235, 0.087 and 0.012 respectively. From these ratios we can say that the above four factors can be ordered in terms of their relative importance as: control structure, program size, data structure and execution structure complexities.

CHAPTER 3

EXPERIMENTS ON PROGRAM COMPREHENSION WITH NOVICE PROGRAMMERS

The main finding of the experiment described in Chapter 2 is that the four factors, viz., program size, control structure, data structure, and execution structure complexities contribute independently to the perception of overall program complexity. This result defines a boundary for further experiments. Since the results of the previous chapter were based on judgemental data, it appeared proper to study the effect of these four factors on actual program comprehension.

The organization of this chapter is as follows. The first section discusses different measures of program understanding, the second section contains discussion about the suitability of different program complexity metrices, and the last three sections include the details of the three experiments, analysis of their data and the results.

3.1 MEASUREMENT TECHNIQUES

The program comprehension can be measured by different techniques. Weissman,[Weissman, 73], used three measures of program understanding--hand-simulation of the program, filling in the blanks in a paragraph describing the program and a subjective rating of program

understanding. Shneiderman, [Shneiderman, 76], had discussed other measurement techniques. The program modification task may take a long time and can usually be accomplished with only comprehension of a section of the program. Similarly, location of a bug, response to a set of questions about the values of variables, program's output, etc., can be done only with partial understanding of the program. Shneiderman, [Shneiderman, 76,77], and Sheppard et al., [Sheppard, 78], used ability to memorize the program as a measure of program comprehension. Memorization of a complex program can only be achieved by abstracting meaningful structures at program level or at a higher level of problem domain.

Success at memorization implies that the subject has understood the low level details of each statement, intermediate level grouping and the overall function of the program. Subjects have to apply their knowledge of syntax of programming language in reconstructing the program. The reconstructed program reflects the success of the subjects in conveying the semantics of the program through syntax of the programming language. Communicating through the rigid syntax of the programming language may demand an extra effort on part of the subject. This will be certainly true in the case of beginners, if not for the experienced programmers.

Other limitation of the technique is due to the fact that there is no reasonable basis to decide about the time

to be allotted to the subjects for understanding the program. The learning theory within the framework of network models assumes that lower order cogits of an information structure are acquired, strengthened and associated during learning [Roth, 77]. The process ofestablishing, strengthening and associating representation and then unitizing configurations of associated representations are applied recursively as learning progresses. Both structural representation and the processing of knowledge change qualitatively as learning progress[Roth, 77].

The recall and recognition of partial and complete knowledge structure should vary according to current state of the memory representation of the complete structure. Horowitz et al.,[Horowitz, 72], ha~~s~~ve investigated this problem. The suitability of the recall as a measure of program comprehension is doubtful because of its dependence on time allotted for learning. Recall would prove to be a good measure if sufficient time has been given for unitization of the mental representation [Roth, 77].

In the case of small programs (less than 100 lines) it would be difficult to differentiate between rote memorization and understanding from the reconstructed programs.

Accordingly, we have decided to use both description of program in natural language and program reconstruction as measure of program understanding. This would also allow a comparison of the two measures.

3.2 COMPLEXITY METRICS

A brief survey of various complexity metrics along with their limitations has been given in Chapter 1. In this section, the measures used for selection of programs for experiments are discussed.

A. Program Size:

The number of executable object instructions have been used as a measure of program size [Schick, 78] . The number of object instructions is machine-dependent and subjects, in general, do not use machine-dependent features to understand the source program. Further, this unit of measure as well as Halstead's measure, [Halstead, 75], of program size in terms of operators and operands are at low level compared to unit used in cognitive process. Since we do not have any qualitative or quantitive basis to hypothesize about cognitive unit and their formation process in the context of program comprehension, we are unable to formalise a better measure.

It has been shown that the program size in number of source statements is strongly correlated with the occurrences of actual software errors [Thayer, 75]. In addition, a statement of source program is a unit at higher level than operator and operands. It appeared reasonable to assume a source statement as cognitive unit.

Consequently, we decided to use number of source statements as a measure of program size.

In the experiment, factor of program size had two levels: <u>small</u> and <u>big</u>. The <u>small</u> programs contained 25-30 statements whereas the <u>big</u> programs contained 45-70 statements*.

B. Control Structure Complexity:

There are different metrics for control structure complexity [Ronback, 75],[McCabe, 76], [Woodward, 79]. These measures as pointed out earlier, do not reflect the overall psychological complexity of the control structure. The measures are based on the numerosity of certain features of the control structure such as number of basic paths, number of 'knots', number of levels, etc. The numerosity affects the interpretation process but not the learning process [Löfgren, 77,78]. The learning is affected by pattern or regularity in the object to be learned. None of these measures capture this aspect of program comprehension. So in our opinion all the above measures are of same class, may be with some quantitative difference in their

---

* We are concious of the fact that the result obtained with above classification of size may not generalize to larger programs. However, these results will certainly be applicable at subroutine or module levels which form the basis of all reliable software products.

suitability. Further, there is no quantitative evidence about correlation between cognitive load and these metrics.

Since we did not wish to add one more metric on the basis of subjective arguments, we decided to use the count of control statements such as IF's and DO's in a program, as proposed by Gilb, [Gilb, 77], as a measure of control structure complexity for selection of programs for the experiments. Gilb has proposed this measure on the basis of results of [Amster, 76].

C. Data Structure:

To the best of our knowledge, we do not know of any metric proposed for data structure. This is partially due to late realisation of the importance of data structures in program development and understanding. The data structure plays an important role in building the mental representation of the program and in learning.

The structural property of data is due to relation between their elements. The relation between data elements may be formally abstracted in terms of predicates and axioms. But defining the complexity in terms of number of predicates does not appear very sound, because the semantic complexity of each predicate or axiom is not same. Further, it will be even more difficult to define the complexity of user-defined data structures. The user may choose arbitrary number of

axioms or predicates to define data structures. In addition, the predicates or axioms are not the cognitive units to understand data structures.

The other alternative is to define complexity in terms of some property of storage structure or access mechanism. However, this leads to a machine-dependent measure.

In cognitive process, names play an important role. The structure of mental representation is expected to be defined in terms of these names or objects of problem domain which they represent. So it seemed proper to define data structure complexity in terms of number of variable names needed to define a data structure. For example, in the case of array A[i] two variable names are used.

The language used in experiment was FORTRAN which has very limited data structure constructs. The criterion used for selection of programs for the experiments was that each variable contribute one unit and each vector and matrix contribute 2 and 3 units to data structure complexity.

D. Execution Structure Complexity:

The execution structure of the program may be characterized by execution of sequence of assignment statements in the program. The regularity of the execution of sequence of assignment statement is represented by control structure of the program. The only aspect

left out is numerosity of the assignments in the program and the execution structure of the assignments. Since the execution structure of assignment statements is machine-dependent feature and is not expected to play any major role in cognition process, the complexity measure should reflect the two aspects: the number of assignment statements and the number of operands needed for each assignment. There was no quantitative evidence to choose a measure. We decided to use a nominal scale on the basis of number of assignment statements in the program for selection of programs for the experiments. A preliminary experiment was also conducted to investigate factors contributing to execution complexity of assignment statements.

3.3 EXPERIMENT 1 : EXPRESSION EVALUATION

The main purpose of the present experiment was to study the effects of number of operands in an (FORTRAN) assignment statement and its meaningfulness in some problem domain on its executional complexity. A source statement was selected as unit for investigation because it is expected that this is the unit used in cognitive process.

A. Subject:

One hundred and five students enrolled in an introductory programming course (TA-306) at the Indian Institute of Technology, Kanpur, served as subjects.

These students had started writing short (less than 100 lines) programs as their laboratory and class assignments. The language taught in the course was FORTRAN. The experiment was conducted as a 20-minute quiz in the middle of the semester. The weighted quiz marks formed part of their final grade.

B. Quiz Paper:

The quiz paper was single page cyclostyled sheet, containing the instruction to the student and ten FORTRAN assignment statements. The students were asked to find values of variables after execution of assignment statements. They were not permitted to use calculators. They could write the answer in simplified fraction. They were also asked to mark any error found and calculate the value after correction. Errors of mixed mode were introduced purposely to pursuade the students to read and understand each statement carefully.

There were ten assignment statements in the quiz. The first five statements formed the group of meaningful statements and the rest five the group of meaningless statements. The meaningfulness of the statements were due to their direct relevance to some problem domain and partially due to selection of variable names which were suggestive of their meaning and purpose. First statement was a formula from mechanics; the second was unit conversion formula from Centigrade to Farenhiet and so on. The meaninglessness was due to the absence of relevance of statements to any problem domain. The

names of variables in this group were not natural, that is, they did not suggest any meaning.

The ten statements were classified in three groups on the basis of number of operands in them. The quiz paper is reproduced in Appendix C.

C. Procedure:

The quiz was conducted in a big class on a scheduled time. The instructor explained the instruction verbally to the students. They were requested to seek any clarification about the quiz. After this initial rapport, the quiz paper was distributed. The students were instructed to write their name and roll number on the sheet. They were reminded of the allotted time for the quiz. After 20 minutes, papers were collected.

The quiz was evaluated on a 10-point scale. The grader had extensive experience in grading student papers. The answers written in wrong mode were graded incorrect.

D. Results:

An analysis of variance was performed on the data [Winer, 71, p. 105]. The difference between mean scores for meaningful - and meaningless - group was significant, $F(1, 104) = 4.826$, $p < 0.05$. The mean score for the meaningful - group ($M = 3.18$) was higher than the mean score for the meaningless - group ($M = 2.88$). This suggests that

evaluation of meaningful expression was easier than that of meaningless ones.

Similar analysis of variance was performed on data to investigate the effect of number of operands on evaluation of expressions. There were three groups of expression on the basis of number of operands. The difference between the mean score for the first group (number of operands = 4) and the second group (number of operands = 5) was statistically significant, $F(1, 208) = 57.29$, $p < 0.05$. Similarly, the difference between mean score for the first group and the third group (number of operands = 6) was also statistically significant, $F(1, 208) = 34.996$, $p < 0.05$. However, the difference between mean scores of the second and third group was not significant, $F(1, 208) = 2.733$. The mean score for the three groups were 7.76, 5.38, 5.90, respectively.

These results suggest that the expression evaluation becomes increasingly difficult as the number of operands increase. This result confirms our general expectation about complexity of expression evaluation. However, it is important to note that this effect is significant even in the case of statements having a few operands as 4,5 and 6.

3.4 EXPERIMENT 2 : DATA STRUCTURE COMPLEXITY

This experiment was conducted to study the effect of data structures and meaningfulness of the program on program understanding. Data structures can be expected to

affect the complexity of the programs in two ways. First, the numerosity and syntactic complexity of data would surely increase the complexity of the program. For example, a program having ten variables is more complex than one having just two variables. Similarly, a program with many 3-dimensional arrays is more complex than one with few vectors only. Secondly, complexity is also affected by whether the data names and organization chosen are "natural" to the problem domain. The selection of "natural" names and organization of data will facilitate understanding due to referential meaning of data names in the context of some problem domain. An example of "natural organization" is representation of a chess board as matrix. This second aspect is responsible for meaningfulness of the program.

A. Subjects:

One hundred and five students of an introductory programming course (TA-306) were selected as subjects. They were randomly divided into five groups, each consisting of 21 subjects. As in Experiment 1, the language used in the course was FORTRAN. The experiment was conducted as 1-hour quiz, 15 days before the end semester examination. Their scores in the quiz were added to the final grade.

B. Quiz Paper:

The quiz paper contained one of the five programs. The first program did not have any matrix or vector. It has 14 simple variables. The variable names were selected

in such a way that they did not carry any positive or negative indication of their relevance to any problem domain. All names were framed by randomly choosing two characters from a set of alphabets. The assignment statements and decision structures were also meaningless. That is, the assignment statements were constructed by randomly choosing variables and operators, and they did not have any relevance to a problem domain. The decision structures were framed in a similar manner. This program was thus a collection of valid FORTRAN statements without any analogy with a problem domain.

The second program was a purposeful program. The program computed average percentage of seats filled up in each section of a course and it also computed average number of section per course in a department. The program did not have any vector or matrix. It had 15 simple variables. The chosen variable names were neutral. They did not provide any hint of their relation to a problem domain. All variables had two characters selected randomly from the set of alphabets.

The third program was same as the second except the variable names. In this program, the variable names were of six characters and they were selected in such a way that names suggested their meanings and association with problem domain. For example, ENROLD represented the number of

students enrolled in a section of a course and MAXIM represented the maximum capacity of the section.

The fourth program did the analysis of student's bio-data. After validating some of the input fields, it calculated the number of students, the number of males and females, and their percentage, average age of students, etc. Finally, it printed out complete statistics. The program had eleven simple variables and five arrays. The variables and arrays names were of two characters and they were neutral as in the first and second programs.

The fifth program was another version of the fourth program with only difference in variable names. All variables and vector names were suggestive of their meaning and association with the problem domain.

In all the programs mentioned above, the numerosity of data structures and meaningfulness of the program were varied. Meaningfulness was due to variable names and familiarity of the subject with problems. The first program served as control for meaningfulness of the program. The fifth program was expected to be most meaningful compared to all other programs.

Each program has one line comment in the beginning to set a frame of reference for the subject.

C. Procedure:

The quiz was conducted in a big examination hall. The instructor described the purpose of the quiz and time slots allotted for different task. Cyclostyled sheets of instruction were distributed to them. Students were encouraged to seek clarification in case of any doubt. After this preparation, programs were distributed to them. They were given 25 minutes for reading and understanding the program. Blank portion of the program sheet was allowed to be used as scratch-pad for preliminary calculations. Programs were collected after completion of the allotted time.

In next 10 minutes, they were required to write the description of the program in English. The description would include what program does and how it does. In the last 10 minutes, they were asked to reconstruct the program. It was not essential to use same variable names, statement numbers and formats in the reconstructed program as long as logic of the program remained similar.

They were also required to mention the error found in the program. Some of the errors were intentional and some of them were undetected at the time of testing. It was expected that location of bugs in the program would facilitate the comprehension. The presence of error was indicated in the instruction sheet.

The quiz was evaluated by an experienced grader. Both the program description and the reconstructed program were graded on a 10-point scale. The reconstructed programs were graded subjectively on the basis of logical consistency of the statements reconstructed. The instruction and programs are reproduced in Appendix D and E respectively.

D. Results:

Separate analysis was performed on scores of program description and reconstructed programs. F ratios for the two measures were $F(4,100) = 7.85$ and 11.172, $\underline{p} < 0.05$ respectively. Since the results were statistically significant, the hypothesis of equal mean score for all the five programs was contradicted.

To study the effects of numerosity of data structure in program and meaningfulness of programs on program understanding, all possible comparisons between total scores of each program from program descriptions were made. Total scores of all the programs for both measures (program description and reconstruction) are given in Table 3.1. F ratios for all comparisons of total scores for program description are inclued in Table 3.2 along with sum of squares and MS_{error}.

TABLE 3.1: SUM OF SCORES OF PROGRAM DESCRIPTION AND RECONSTRUCTION

	A	B	C	D	E
Program Description	108	33	71	75	122
Program Reconstruction	150	56	69	67	101

TABLE 3.2

F RATIOS OF COMPARISONS - PROGRAM DESCRIPTION

S.No.	Comparison between programs	SS_C	MS_{Error}	F Ratio
1	A - B	133.928	7.35	18.22*
2	A - C	32.595	"	4.434*
3	A - D	25.928	"	3.527
4	A - E	4.66	"	0.634
5	B - C	34.38	"	4.67*
6	B - D	42.0	"	5.714*
7	B - E	188.59	"	25.65*
8	C - D	0.381	"	0.052
9	C - E	61.928	"	8.426*
10	D - E	52.595	"	7.155*

* indicates statistically significant.

Differences in total scores for program description between programs A and D, A and E, and C and D were not statistically significant. Program D and E had more complex data structures compared to program A, B and C. This suggests that the numerosity of data structures did not hinder the program understanding. The total scores of program D and E were more than those of B and C. This further justifies the ineffectiveness of the numerosity of data structures. The score of program A was more than all the programs except E. This result as well as the ineffectiveness of numerosity of data structures are contrary to our intuitive expectation.

The scores of program C and E were more than those of B and D respectively. Differences of scores between programs B and C, D and E, C and E, and B and D were statistically significant. These significant differences suggest that meaningful programs were easier ~~easire~~ to understand. The meaningful data names and organization facilitate understanding. However, we do not have reasonable argument for higher scores of program A compared to programs B, C and D and so it does not seem proper to generalize the above result.

Similar comparisons were made between total scores of each reconstructed program. Table 3.3 lists F ratios for all comparisons. The total scores of programs B and C were less compared to the scores of programs D and E. The difference between scores of programs C and D was not

TABLE 3.3

F RATIOS OF COMPARISONS - PROGRAM RECONSTRUCTION

S.No.	Comparison between programs	SS_C	MS_{Error}	F Ratio
1	A - B	210.381	6.22	33.823*
2	A - C	156.214	"	25.115*
3	A - D	164,02	"	26.370*
4	A - E	57.166	"	9.190*
5	B - C	4.024	"	0.647
6	B - D	2.88	"	0.463
7	B - E	48.214	"	7.751*
8	C - D	0.095	"	0.015
9	C - E	24.38	"	3.92
10	D - E	27.52	"	4.425*

* indicates statistically significant.

significant. These results supplement the conclusion from the other measure that complex data structure did not hinder the program understanding.

The differences between the total scores of programs B and C was not significant, whereas between D and E was significant. The latter result supports the conclusion that meaningful data names enhance understanding whereas the former one contradicts it. In addition, the score of program A was more than score of all the remaining programs. Since the conclusions are contradicting, it will not be proper to generalize the result.

To compare the two measures of program understanding used in this experiment, a separate analysis of variance was performed on data. The result was not significant, $F(1,104) = 2.83$ and so it may be concluded that both the measures are interchangeable.

A two-way, 5x2, analysis of variance was performed [Winer, 71, p.302], to supplement the above analysis and to study the interaction between programs and two measures. The two factors were program types and measurement techniques, with repeated measure on the second factor.

The F ratios for factor of program type was significant $F(4,100) = 9.19$, $\underline{p} < 0.05$. This supports the findings of earlier analysis. It is interesting to note that factor of measurement techniques did produce marginally significant

difference, $F(1, 100) = 3.90$, $p < 0.06$. It implies that the two measures of program understanding are not the same. This is in contradiction with results of previous analysis.

The interaction between the two factors was significant, $F(4, 100) = 10.8$, $p < 0.05$. Figure 3.1 shows the plots of mean scores of different programs as a function of different measurement techniques. It is evident from Figure 3.1 that program description has a higher scores compared to program reconstruction for meaningful programs. The differences in the scores increase as the meaningfulness increases with one exception. In the case of program A, the reconstruction score is higher than the description score. This may be due to difficulty in learning the meaningless computation. The subjects could not discover any structure and analogy with a problem domain in the process of learning. They found it difficult to describe the program briefly in English. In reconstruction, they probably used rote memorization and they did better.

In summary, the numerosity of data structure in the program did not hinder understanding whereas the meaningful data names and familiarity of problem facilitated understanding. The effectiveness of the program reconstruction and description measures are function of nature of the programs.

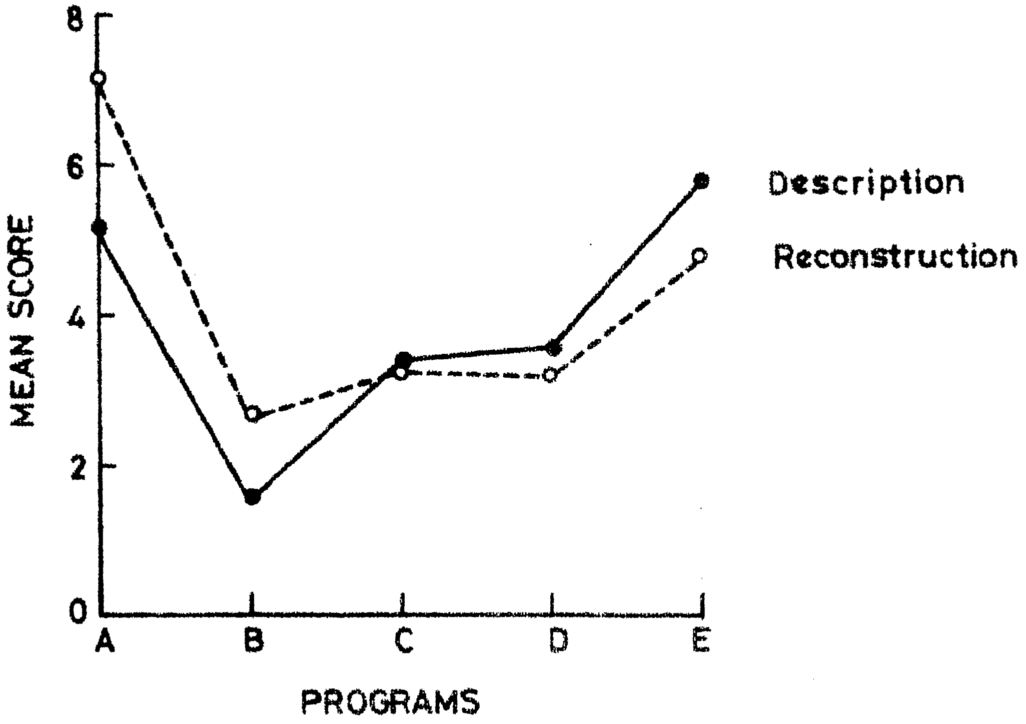


FIG.3.1 COMPARISON OF TWO MEASURES OF UNDERSTANDING

Since, the experiment did not prove the difference between two measures conclusively, we decided to use program reconstruction as measure. We also decided to use only uniformally meaningful programs for further experiments.

3.5 EXPERIMENT 3 : CONTROL STRUCTURE AND EXECUTION STRUCTURE

In the earlier two experiments, only one factor was considered for study at a time. In Experiment 3, our main aim was to study the effect of both control structure and execution structure on program understanding.

A. Design:

A 3x3, control structure x execution structure, factorial design was employed. Each factor had three levels -- low, moderately complex, and very complex. Nine programs ($3^2=9$) were constructed corresponding to each combination of levels of both factors. The levels of control structure, as pointed out earlier, were controlled by number of control statements such as IF's, DO's, and computed GO TO's in the program. The number of IF statements were 0-2, 3-5, and 6-8 corresponding to low, moderately complex, and very complex levels of this factor. For execution structure, the number of assignment statements and the number of operands in them were used as controlling factors. All the nine FORTRAN programs were purposeful and variable names were meaningful. Factors of size and data structure complexity were held constant. These programs are listed in Appendix F.

B. Subjects:

The subjects were one hundred and seventeen students from the same population as in Experiments 1 and 2. They were randomly divided in nine groups, each consisting of 13 subjects. The experiment ws conducted as a 30-minute quiz a week prior to the end semester examination.

C. Programs:

Nine programs used in the experiment are listed in Appendix F. All programs were of 15-20 lines in length. One of the program had an array; others had only simple variables.

D. Procedure:

The quiz was conducted in two big class rooms during one scheduled lecture hour. The instructor had discribed the purpose of the quiz in the previous lecture. The quiz was divided in two parts. In Part A, subjects were required to read and understand the program in 15 minutes. To facilitate their understanding, they were asked to evaluate values of variables at different points in the program on the lines of Weissman, [Weissman, 73]. The first section of the part A paper contained the instruction for this part. The program formed the second section. The last section had the list of variables and line number of programs for which values were to be evaluated. The blank spaces were left for answers. After completion of Part A, the programs were collected.

In Part B, the subjects were required to reconstruct the program which they had read in Part A. The time allotted for this task was 15 minutes. It was not essential to use same variable names and statement numbers as long as logic of the program remained same. Separate instruction was given for this part. Both sets of instruction are given in Appendix F. Students were also requested to point out any error in the program while reconstructing it.

The reconstructed programs were graded on a 100-point scale by two methods. In one method, the number of lines perfectly recalled was used as measure. In the other method, the reconstructed programs were graded subjectively. In this method of grading, logical consistency of the statements recalled was taken into consideration. For example, if there was a statement, GO TO 30, in the middle of the original program and in reconstructed program subjects had perfectly recalled the statement but had put it in wrong position then in the second method this was marked wrong. The method become subjective because logical consistency and related judgement about the extent of understanding will be done by the grader.

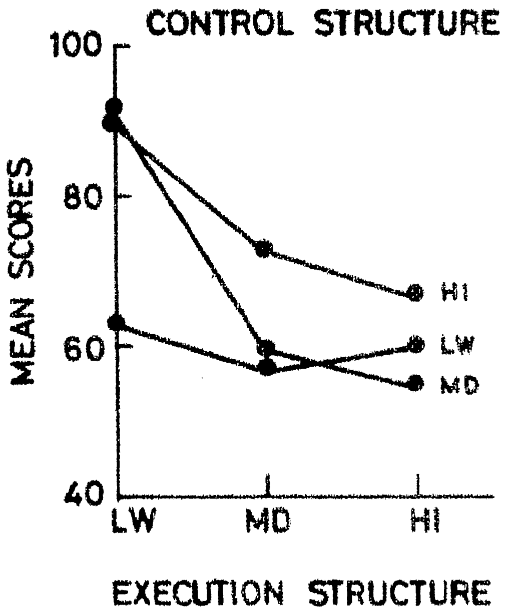
E. Results:

An analysis of variance was performed on data obtained from both the methods of grading. Effects of both control structure and execution structure complexities were

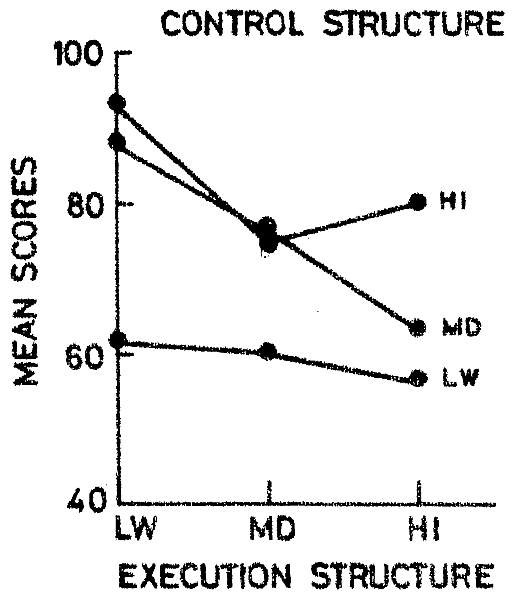
statistically significant, $F(2, 108) = 9.18, 3.52, p < 0.05$. However, the interaction was not significant, $F(4, 108) = 0.97$. Similar result was obtained for the other method of grading (subjective).

The absence of interaction between two factors confirms the judgemental result reported in Chapter 2. Mean scores for programs were plotted as function of control structure and execution structure. The two sets of curves in Figure 3.2 are for the two methods of grading, respectively. It is evident from the curves that as the execution structure complexity increases the mean scores decreases for all levels of control structure. There is, however, one exception. This may be due to noise in data. However, it is interesting to note that mean score increases with the increase of control structure complexity. This is in contradiction with our intuitive expectation.

This unexpected result may be attributed to efficient strategy in the learning process. Two widely accepted meta-inferential techniques are inductive inference and inference by analogy [Bobrow,75a, p.20]. Inductive inference uses a set of facts to form the basis for a general rule for expressing relations. In inference by analogy, if certain criteria of similarity are met between two situations, then a result that pertains to the first situation can be assumed to pertain to the second situation. We do not know which of the rules are used in the learning process.



A. SUBJECTIVE GRADING



B. GRADING BY LINE RECALLED

FIG. 3.2 CONTROL AND EXECUTION STRUCTURE

A closer observation of programs for <u>moderately complex</u> and <u>very complex</u> control structure, however, reveals that there was a syntactic/semantic regularity in the control logic of the program. For example, the two statements,

```
IF(JBIT1.EQ.1) JSUM = JSUM+1   and
IF(JBIT2.EQ.1) JSUM = JSUM+2
```

have a syntactic and semantic regularity. The difficulty in learning process is reduced due to this regularity because the subjects can easily use the two meta-inferential techniques discussed above. This explains the reason for higher score for increasing control structure complexity.

This result has vindicated our view on complexity metrics of control structure. The numerosity of control statements or of any feature of control flow will not reflect the psychological complexity of control structure. Metric has to take into account the effect of regularity on learning.

CHAPTER 4

EXPERIMENTS ON PROGRAM COMPREHENSION WITH EXPERIENCED PROGRAMMERS

The experiments described in Chapter 3 were single-factor ones. The third experiment in that chapter was a factorial experiment, but it considered only two factors. These experiments did not permit the study of interaction effects. Furthermore, the subjects in all those experiments were beginner programmers. The class-room environment and strict time schedule for course prevented us from designing multifactor experiments and using large programs in those experiments. Thus the results from the previous experiments are only partially complete.

This chapter describes two more experiments. The first is a four factor experiments having program size, control structure, data structure, and execution structure complexities as factors. The second experiment studied the effectiveness of the subjective rating of the program complexities. In both these experiments, subjects were programmers with at least 2 years of experience.

First two sections of this chapter contain the details of the two experiments, analysis of data and their results. The last section includes the conclusions drawn from the experiments reported in Chapters 3 and 4.

4.1 EXPERIMENT 1 : FOUR-FACTOR OF COMPLEXITY

The purpose of the present experiment was to study the effect of factors of size, control structure, data structure and execution structure complexities on program comprehension.

A. Design:

A 2x2x2x2 factorial design was used. Each factor had two levels. The decision to have only two levels for each factor was taken due to time constraint with programmers. Programs were selected for each combination of levels of all the factors. Each subject worked on all the sixteen programs.

B. Subjects:

Eight system programmers served as subjects. The same group of programmers had volunteered as subjects in the experiment described in Chapter 2. The characterizing features of the group are repeated here for convenience. The experience of programmers ranged from 2 to 10 years. They had a thorough experience in FORTRAN and Assembly language programming. They also had a good knowledge of COBOL and other languages.

C. Programs:

Sixteen programs were selected from different text books of programming. The programs were modified to meet the criteria of selection. The guidelines for selection

of programs were metrics discussed in section 3.2. Program of length 25-30 and 45-70 statements were selected as the two levels for the factor of size. Programs having 3-5 and 8-10 logical IF or DO statements represented the two levels of the factor of control structure. No distinction was made between the above two control constructs for the sake of deciding levels. In the case of data structures, the number of vectors and matrices corresponding to two levels were 0-3 and 5-7. Similarly, for execution structure complexity, 5-6 assignment statements constituted the lower level. For higher level, the number of assignment statements were 1-12.

Each program had two comments. The first comment was for subjects to fix up their frame of reference. The second comment included the specification and reference in the coded form for the experimenter. The programs were tested for few sets of inputs for each. As the tests were not exhaustive, some errors remain undetected. Some other errors were introduced intentionally. It was thought that the presence of bugs will help to keep reader alert and thus aid understanding. This is particularly important because the subjects worked as user consultants at the computer center for at least four hours a day where they read mostly programs with bugs.

The sixteen programs were from different application area. These programs are reproduced in Appendix H.

D. Procedures:

The experiment was conducted in four sessions, each of 2-hour duration. These four sessions were conducted on four different days. The experiment was conducted in a class room with all programmers simultaneously.

The experimenter described the purpose of the experiment verbally to the subjects. Cyclostyled sheets of instruction were also distributed to them. They were requested to seek clarification in case of any doubt. The instruction is reproduced in Appendix G.

Each session was divided in four subsessions, each of 30-minute duration. In each subsession, programmers received one program selected randomly. Each programmer received the same program. There were two reasons for not randomizing the selection of programs between programmers. First reason was that the programs were of different types. It was expected there would not be any learning effect from their order of presentation. The second reason was that the subjects could discuss each other program after the subsession. This discussion was not desirable for the purpose of experiment.

The programmers were urged to read and understand the program. Programs were collected after completion of 15 minutes. In the next 15 minutes, they wrote the comments

and reconstructed the programs. In their comments, they were asked to include a brief summary of the program, any error found in the program and their general opinion about the complexity of programs. A period of five minutes was given to the above task. It was expected that the exercise in this five minutes would make the features of the program clearer to the subjects and the subjects would be able to organize their idea properly and would do better in program reconstruction. In the last ten minutes, subjects were required to reconstruct the program. It was not essential to use the same variable names, statement numbers and formats. A break of 15 minutes was given after two subsessions. In this way, the experiment was conducted for four programs on a particular day.

The reconstructed programs were graded on a 10-point scale by an experienced grader. The grading was done subjectively. That is, the logical consistency of the recalled statements was the major criterion and the judgement about the extent of understanding was subjective. The comments, written by programmers, also helped in this grading. The few syntax errors that were found did not introduce any ambiguity as to the intention of the writer. These errors were totally ignored.

E. Results:

A 8x2x2x2x2 analysis of variance was performed [Winer, 71, p. 140]. The five factor being subjects,

program size, control structure, data structure, and execution structure complexities. As each subject reconstructed all the programs, interaction of the subject with a source of variance was used as error term in construction of F ratios [Winer, 71, p.202].

The F ratios for all main and interaction effects are given in Table 4.1. All the factors except program size had statistically significant effect on program reconstruction. Three 2-way interactions, program size x control structure, program size x data structure, and data structure x execution structure, were significant. The rest of 2-way interactions were not significant. Three 3-way interactions were significant. The non-significant interaction was program size x control structure x execution structure. The presence of these significant interactions is in contradiction with the judgemental result of Chapter 2.

For graphical analysis, mean scores of reconstructed programs were plotted as function of 2- and 3-factors. The six sets of curves corresponding to all 2-factor combinations are given in Figure 4.1. Similarly, curves for all 3-factors combination are given in Figures 4.2 and 4.3.

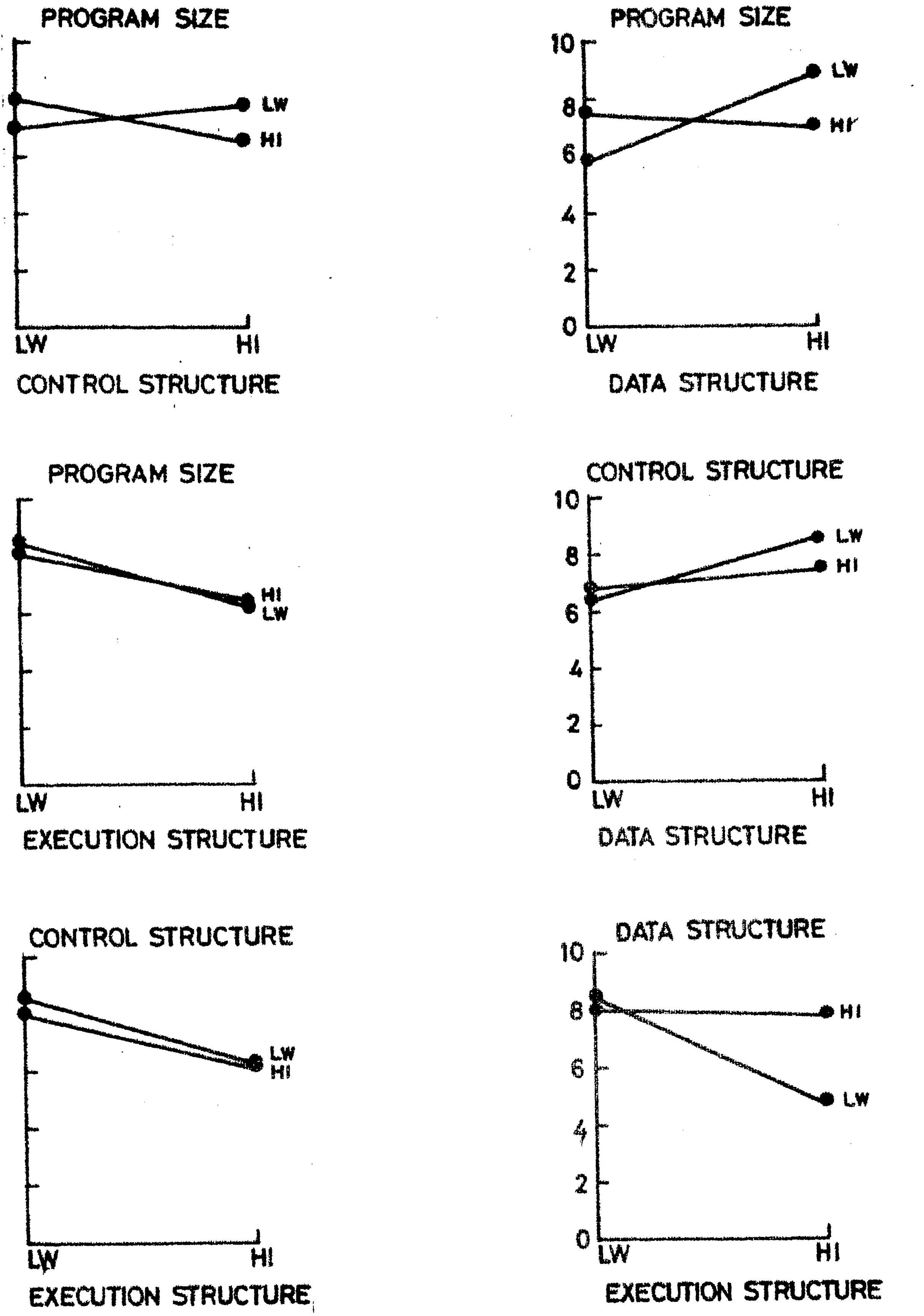
It is evident from the curves of Figure 4.1 that when the program size is <u>big</u> the mean score decreases with the increase of control structure, data structure, and execution structure complexities. Similarly, the increase

TABLE 4.1

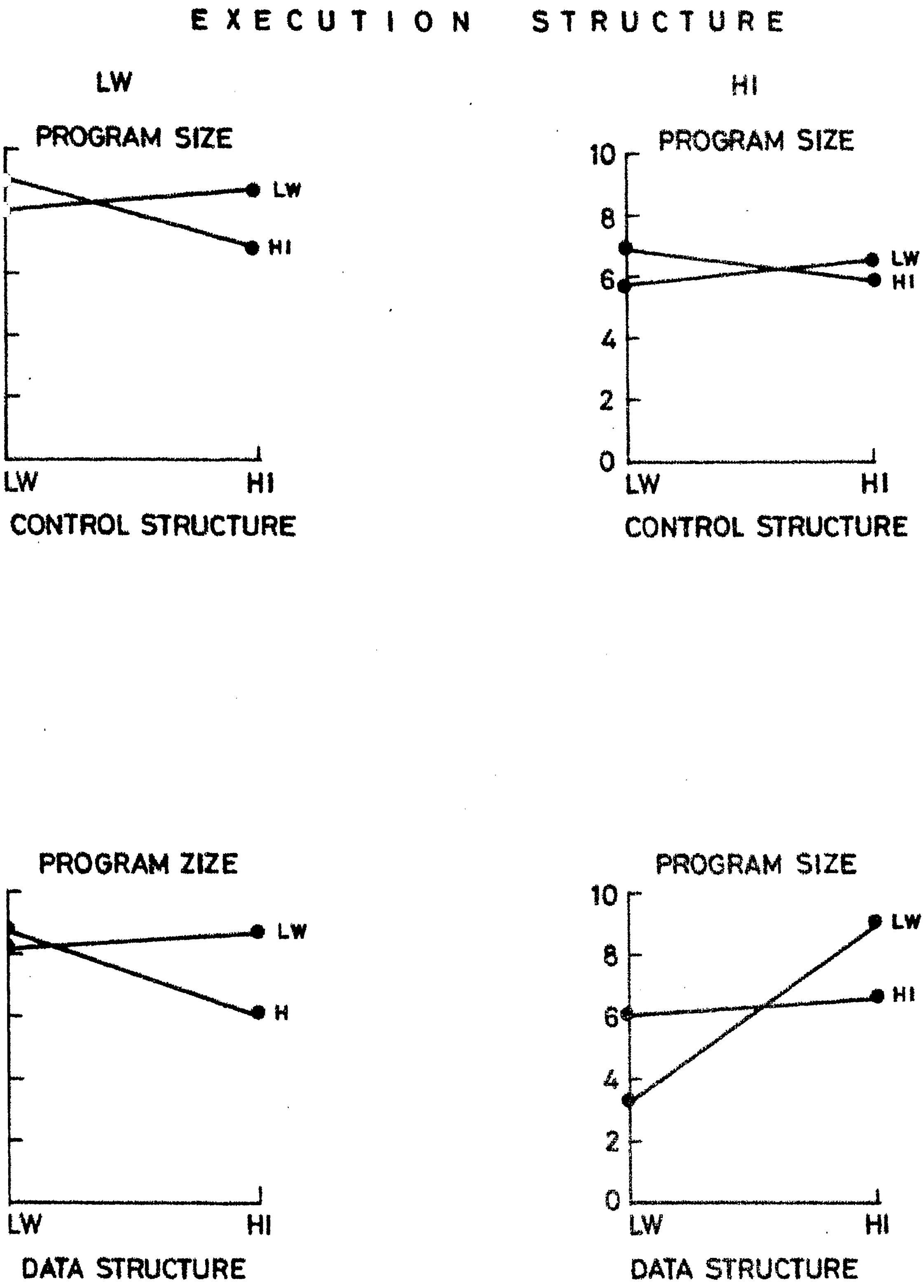
F RATIOS FOR FOUR FACTORS AND INTERACTIONS

S.No.	Factors and Interactions	F Ratio
1	Program size	0.131
2	Control structure	6.88*
3	Data structure	162.00*
4	Execution structure	101.88*
5	Program size x Control structure	6.763*
6	Program size x Data structure	50.159*
7	Program size x Execution structure	1.552
8	Control structure x Data structure	3.706
9	Control structure x Execution structure	0.394
10	Data structure x Execution structure	23.140*
11	Program size x Control structure x Data structure	6.219*
12	Program size x Control structure x Execution structure	0.137
13	Program size x Data structure x Execution structure	9.084*
14	Control structure x Data structure x Execution structure	9.529*
15	Program size x Control structure x Data structure x Execution structure	4.991

* indicates statistically significant.

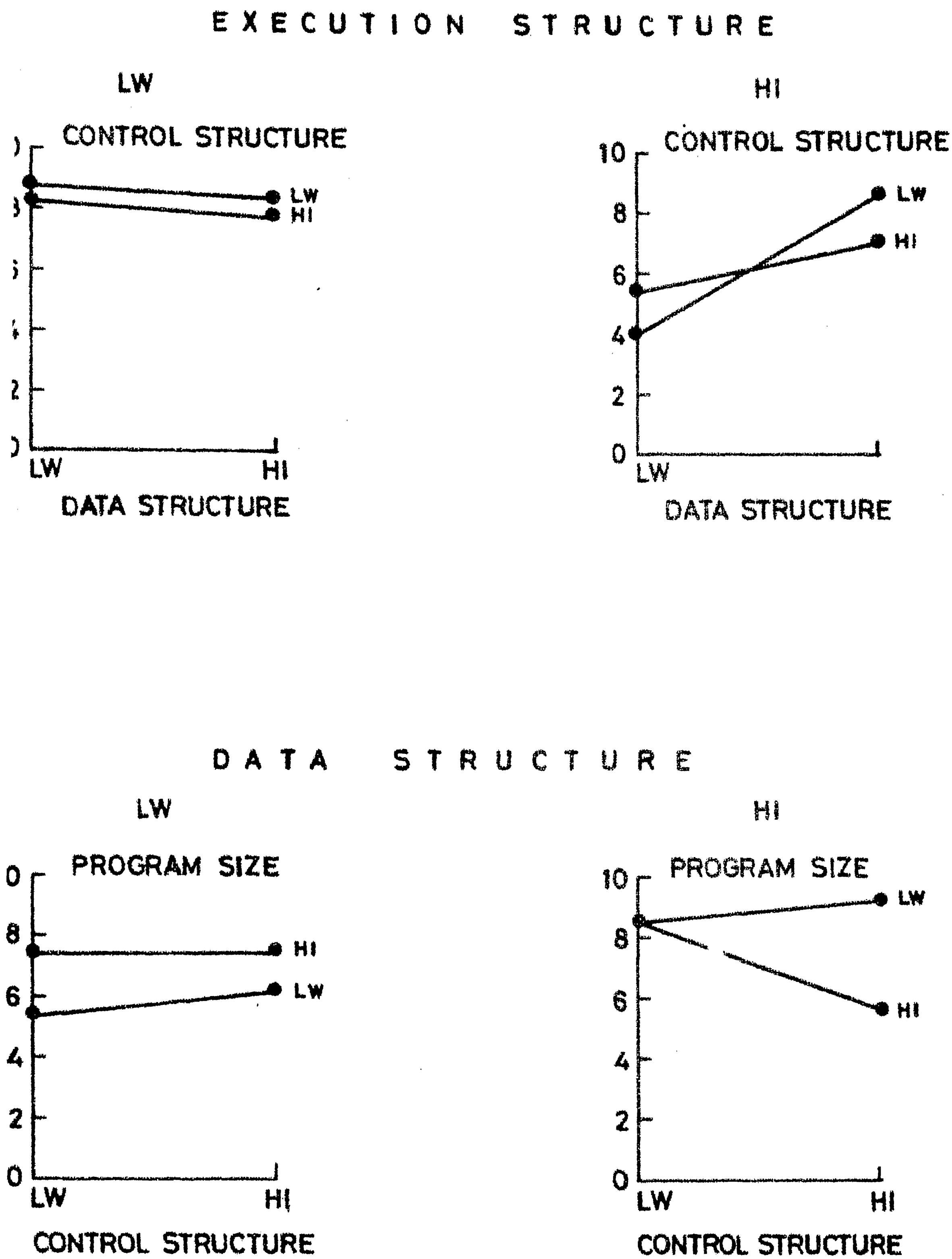


XIS REPRESENTS MEAN SCORES IN PROGRAM RECONSTRUCTION FOR ALL CURVES



; REPRESENTS MEAN SCORE IN PROGRAM RECONSTRUCTION IN ALL CURVES

FIG. 4.2 3-WAY INTERACTIONS OF FOUR FACTORS



Y-AXIS REPRESENTS MEAN SCORE IN PROGRAM RECONSTRUCTION FOR ALL CURVES

FIG. 4.3 3-WAY INTERACTIONS OF FOUR FACTORS

of execution structure complexity decreases the mean score for all the levels of all the factors. However, in the case of program size small, the mean score increases with increase of control structure and data structure complexities. Further, for lower level of control structure complexity, the mean score increases with increase of data structure complexity. On the basis of results from the judgemental experiment, the increase of any of the four factors would decrease the mean score, i.e., increase the overall program complexity. Furthermore, each factor would contribute independently to program complexity. That is, there would not be any interactions. The only set of curves satisfying the above expectation is for control structure x execution structure complexities. The trend of the other curves do not support the above expectation.

The only conclusive result from the above curves is that execution structure complexity affects program understanding and number of assignment statements and the number of operands in them are effective measure of this complexity. For other factors, due to contradicting evidences, no such conclusive results can be inferred.

A closer look on these curves reveals that the deviation from the expected result is at one point in each set of curves. This point corresponds to program size small and lower levels of control structure and data structure. Similarly, for other two sets of curves the

points correspond to low levels of control structure and data structure and low level of data structure and high level of exec.struct.respectively. The comments written by programmers gave enough hint for a possible explantion of this unexpected result.

Six out of eight programmers had commented that the program number 2 (Appendix H, p.169) was difficult to understand. Different vreasons were forwarded by the subjects for the above difficulty in understanding the program. Three of them indicated that variable REQ caused ~~program~~ problem to them. They could not make out what this variable represented. The reasons forwarded by others were insufficient time, presence of errors in the program, and unfamiliarity with the problem. Similarly, the program number 6 (Appendix H, p.175) was also described as difficult. The unanimous reason for the difficulty was switching of variable JW. They also reported difficulty in understanding the program number 10 and 16. In program number 10, it was due to nature of the algorithm; whereas for program 16, it was due to size of the program and unfamiliarity of some of the subjects with statistical formulae.

The reasons forwarded for the difficulties in understanding suggest that these programs were more complex than they were originally assumed to be. Moreover, the increased complexity was not due to the four factors of consideration,but due to some external unexpected factors. Due

to this increased complexity of the programs the mean scores exhibit an unexpected deviation. The same reasoning also explains the anomalies in the curves of Figures 4.2 and 4.3.

In the light of the above explanation, the results can be summarized as the factors of control structure, data structure and execution structure complexities affected the program understanding, whereas the program size did not affect. The factors other than these four had a dominant effect on program understanding. However, these unexpected factors remain unidentified at this stage.

From the above results, it may be concluded that the difference between two levels of the factor of program size was not enough. Even longer programs should have been used for the experiments.

It is evident from the comments about variable REQ and about the familiarity with formulae and algorithms (program 10 and 16) that problem domain knowledge base plays a very important role in program understanding.

4.2 EXPERIMENT 2 : COMPLEXITY RATING

The purpose of the present experiment was to study the plausibility of the arguments forwarded to explain the anomalies in the results of the previous experiment. It was argued that some of the programs (2, 6, 10, 16) were more complex than they were originally thought to be and the source of

complexity was not among the four factors of discussion. It was decided to get the complexity of the programs rated by different group of programmers. The experiment also provided an opportunity to study the effectiveness of the subjective rating of program complexity.

To minimise the subjectivity in the rating, the subjects were required to write the summary of the programs after understanding. Subjects rated the complexity of the programs after writing the summary and then indicated the factors responsible for the complexity as well as their contribution in percentage. It was thought that writing the summary of the program, identifying the sources of complexity, and their percentage contribution would minimise the subjectivity in complexity ratings of the programs. The details of the experiment and results are described in the following sections.

A, Subjects:

Seven senior graduate students of computer science program at the Indian Institute of Technology, Kanpur served as subjects. They had approximately two years of experience in FORTRAN and Assembly language programming. They also knew COBOL.

B. Procedures:

The experiment was conducted in a class room with all subjects together. The experiment was divided in four sessions of 2-hour duration. Each session has four

subsessions of 30-minute duration. The four experimental sessions were scheduled on different days.

The task and purpose of the experiment were explained to the subjects. Instruction sheets, reproduced in Appendix I were also distributed to them. In each subsession, subjects received a different program chosen randomly. All subjects worked with same program in a subsession. They were requested to read and understand the program. After 15 minutes, programs were collected back. They were asked to write a brief summary of the program, which they have just read, in next five minutes. In the last five minutes, subjects judged the complexity of the program. They gave their complexity rating on a 10-point scale. They also estimated the percentage contribution of various factors to the program complexity.

A blank sheet of paper was provided to each subject for writing the summary. For complexity rating and percentage estimation, cyclostyled sheets listing the factors of program complexity and with blank space left for answers, were distributed.

This procedure was repeated over all the subsessions.

C. Results:

A 7x2x2x2x2 analysis of variance was performed on data. The factors were subject, program size, control structure, data structure, and execution structure complexities. The F ratios were computed as described in the previous experiment. Tables 4.2 lists the F ratios for main and interaction effects.

TABLE 4.2

F RATIOS FOR COMPLEXITY RATING

S.No.	Factors and Interactions	F Ratio
1	Program size	1.459
2	Control structure	0.069
3	Data structure	11.413*
4	Execution structure	22.711*
5	Program size x Control structure	12.692*
6	Program size x Data Structure	11.743*
7	Program size x Execution structure	11.172*
8	Control structure x Data structure	15.54*
9	Control structure x Execution structure	10.159*
10	Data structure x Execution structure	5.448*
11	Program size x control structure x Data structure	1.877
12	Program size x Control structure x Execution structure	0.264
13	Program size x Data structure x Execution structure	2.591
14	Control structure x Data structure x Execution structure	39.69*
15	Program size x Control structure x Data structure x Execution structure	8.092*

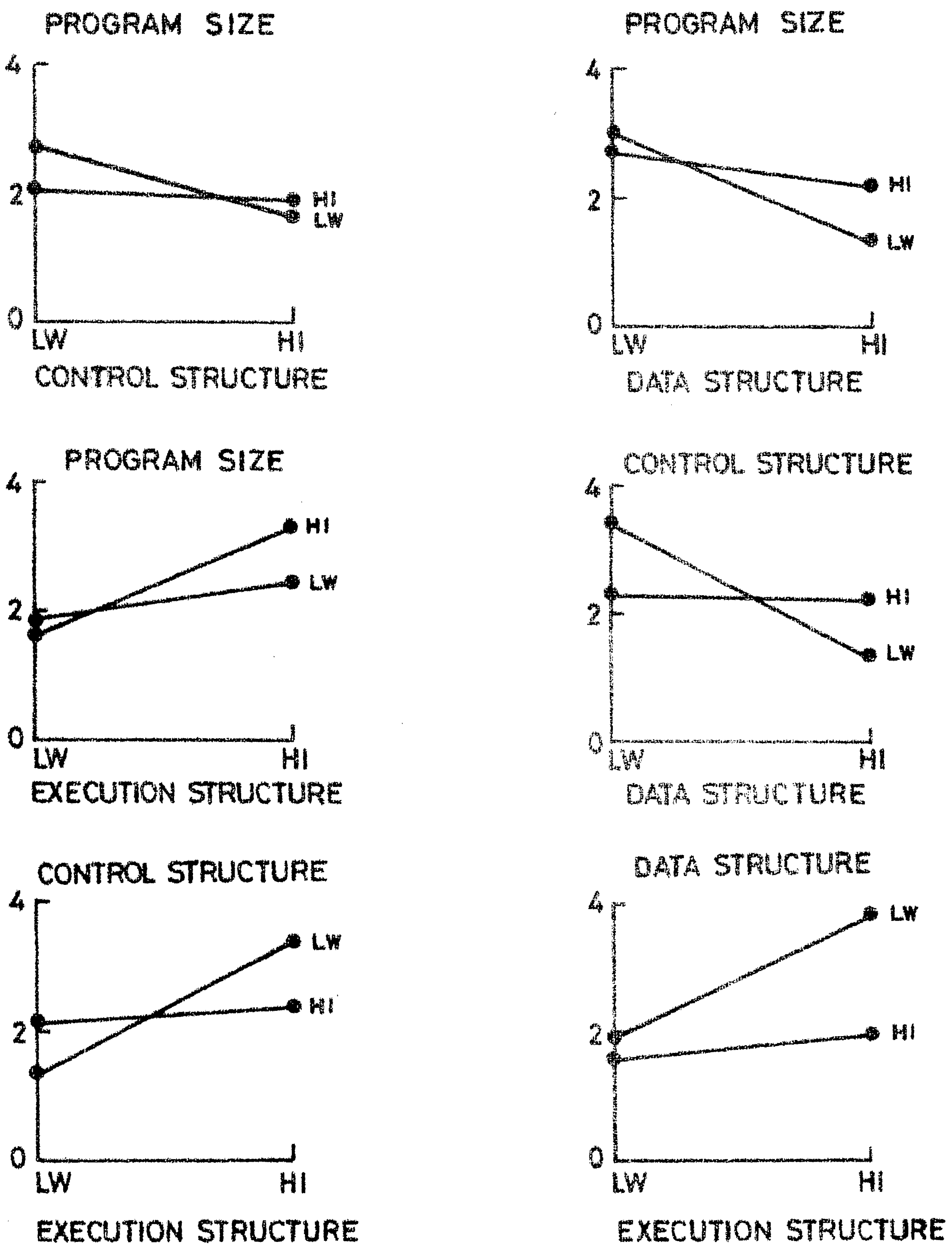
* indicates statistically significant.

For graphical analysis, the mean program complexities were plotted as functions of 2-, 3- and 4-factors. There were six sets of curves for all 2-factor combinations. The curves are given in Figure 4.4. The curves corresponding to 3- and 4-factor combinations are plotted in Figures 4.5, 4.6 and 4.7 respectively.

It is evident from the curves of Figure 4.7 that the program number 2 had highest complexity among all the sixteen programs. The ordering of programs on the basis of these curves in Figure 4.7 is 2, 10, 16, 13, ... The points corresponding these more complex programs are encircled on the curves along with number of the program. Except for the above four programs, all other programs had comparable complexity.

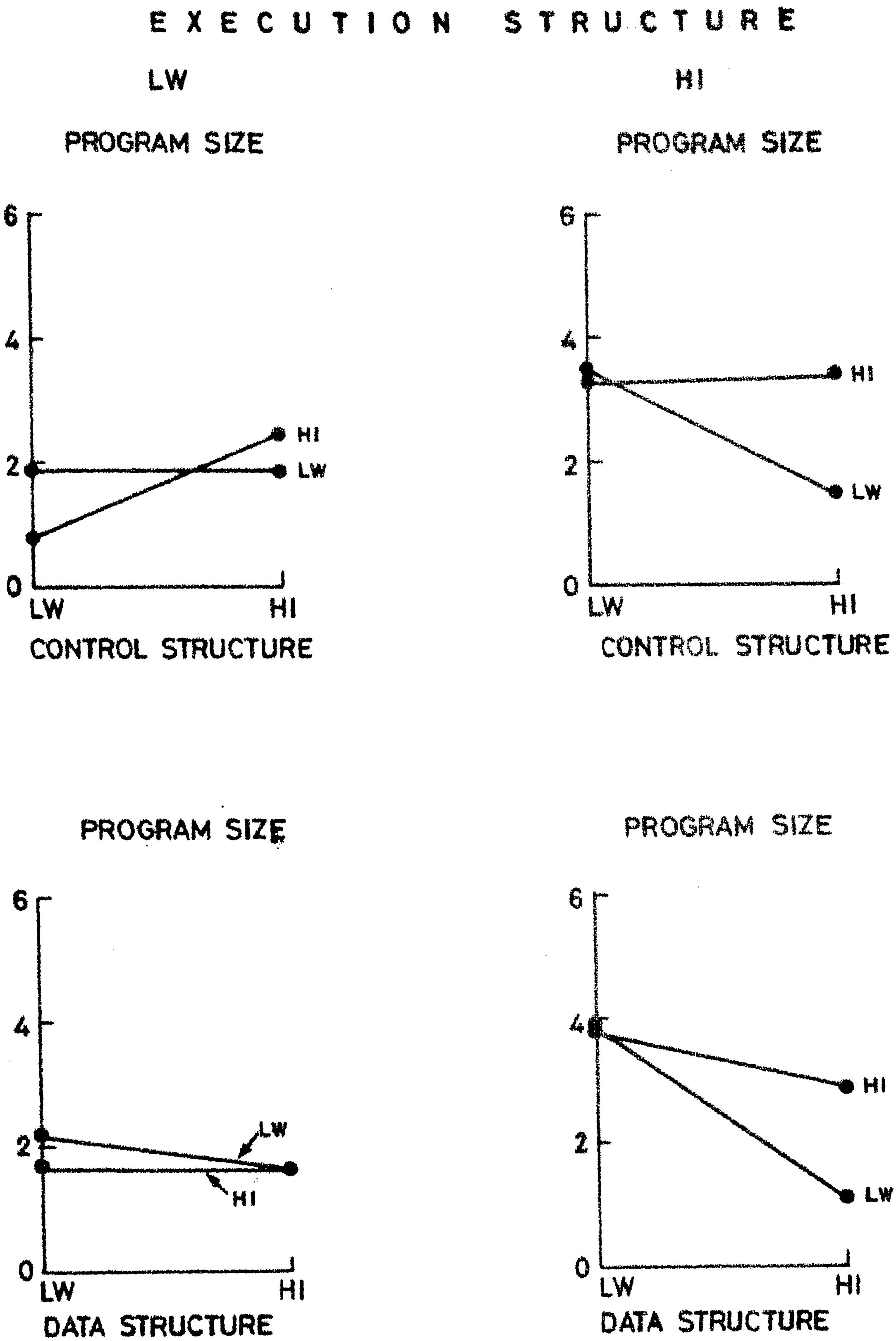
The trend of the curves, in general, supplements the results of the previous experiment. A comparison of Figure 4.1 and 4.4 reveals some interesting similarity. Even though the subjects for the two experiments were drawn from the distinctly different population, the results have a remarkable similarity. Subjects of Experiment 1 had scored higher for those programs which had low complexity ratings from the subjects of the Experiment 2. The difference in the results of the two experiments are quantitative but not qualitative.

The above observation and F ratios suggest two conclusions. First, the argument forwarded to explain



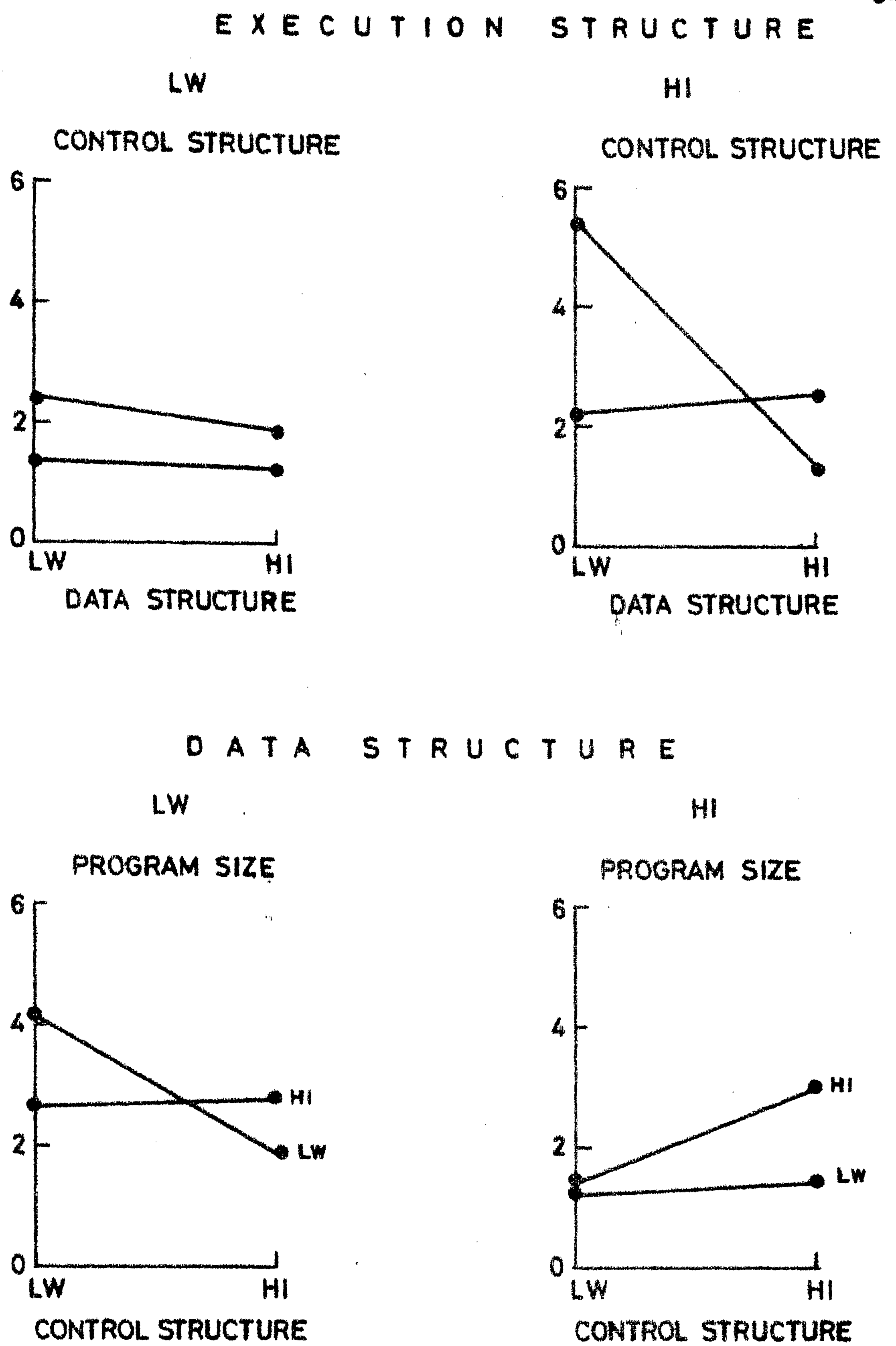
Y-AXIS REPRESENTS MEAN PROGRAM COMPLEXITY FOR ALL CURVES

FIG.4.4 2-WAY INTERACTIONS IN COMPLEXITY RATING



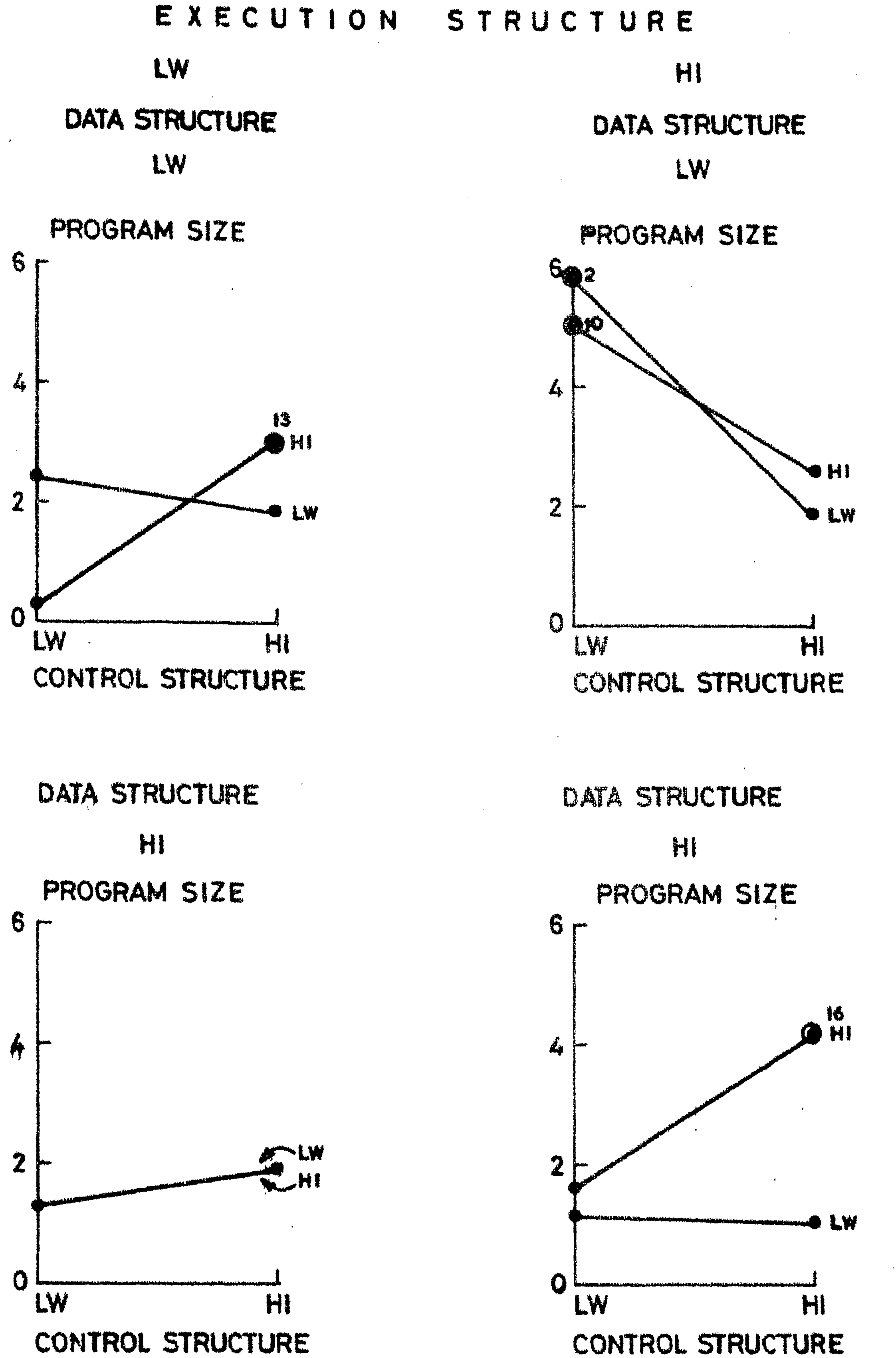
Y-AXIS REPRESENTS MEAN PROGRAM COMPLEXITY FOR ALL CURVES

FIG. 4.5 3-WAY INTERACTIONS IN COPLEXITY RATING



Y-AXIS REPRESENTS MEAN PROGRAM COMPEXITY FOR ALL CURVES

FIG. 4.6 3-WAY INTERACTIONS IN COMPEXITY RATING



Y-AXIS REPRESENTS MEAN PROGAM COMPLEXITY FOR ALL CURVES

FIG. 4.7 4-WAY INTERACTIONS IN COMPEXITY RATING

the results of the previous experiment seems to be valid. Some of the programs were more complex than they were intended to be. Second, the difference in two levels of factor of program size were not enough to affect the complexity. Further, the numerosity of control statements in the program did not increase the program complexity as was expected. The significant factors were data structure and execution structure complexities (see Table 4.2)

In addition, the results of the experiment suggest that subjective rating of complexity of programs may be a good measure of program understanding provided the subjects are given some initial exercise to understand the program.

Encouraged by similarity in the results of two experiment, we decided to analyse the summary of the programs written by the subjects in Experiment 2. Summary was graded on a 10-point scale. A 7x2x2x2x2 analysis of variance was performed on the score from the summary. The F ratios for main and interaction effects are given in Table 4.3.

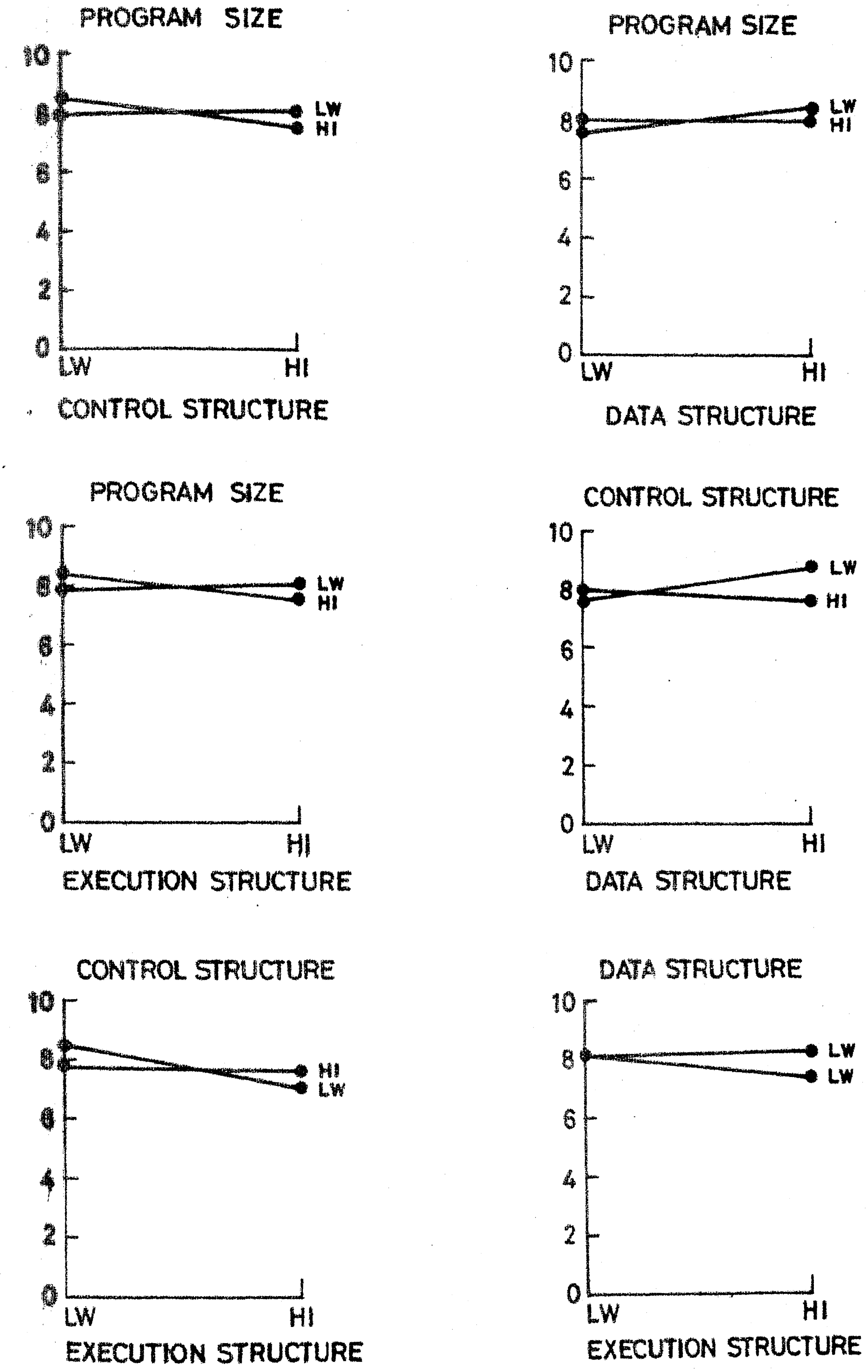
Mean scores for programs were plotted as function of combination of factors. The six sets of curves for all 2-factor combinations are plotted in Figure 4.8. The curves corresponding to 3-factor combinations are given in Figures4.9 and 4.10.

TABLE 4.3

F RATIOS FOR PROGRAM SUMMARY

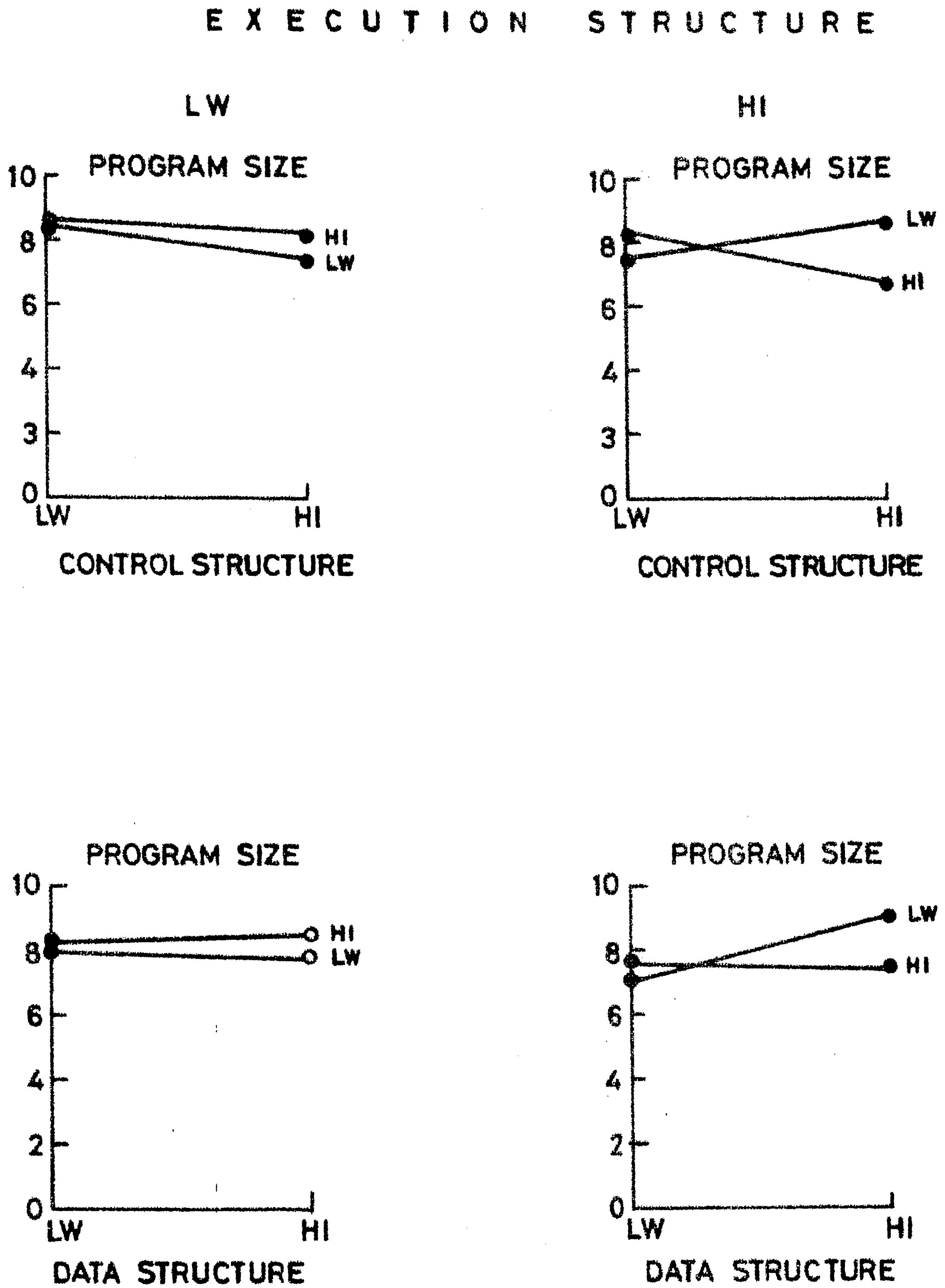
S.No.	Factors and Interactions	F Ratio
1	Program size	0.0357
2	Control structure	1.964
3	Data structure	1.755
4	Execution structure	0.474
5	Program size x Control structure	1.48
6	Program size x Data structure	5.04*
7	Program size x Execution structure	2.245
8	Control structure x Data structure	5.54*
9	Control structure x Execution structure	0.658
10	Data structure x Execution structure	1.638
11	Program size x Control structure x Data structure	0.46
12	Program size x Control structure x Execution structure	12.11*
13	Program size x Data structure x Execution structure	9.33*
14	Control structure x Data structure x Execution structure	0.49
15	Program size x Control structure x Data structure x Execution structure	3.06

* indicates statistically significant.



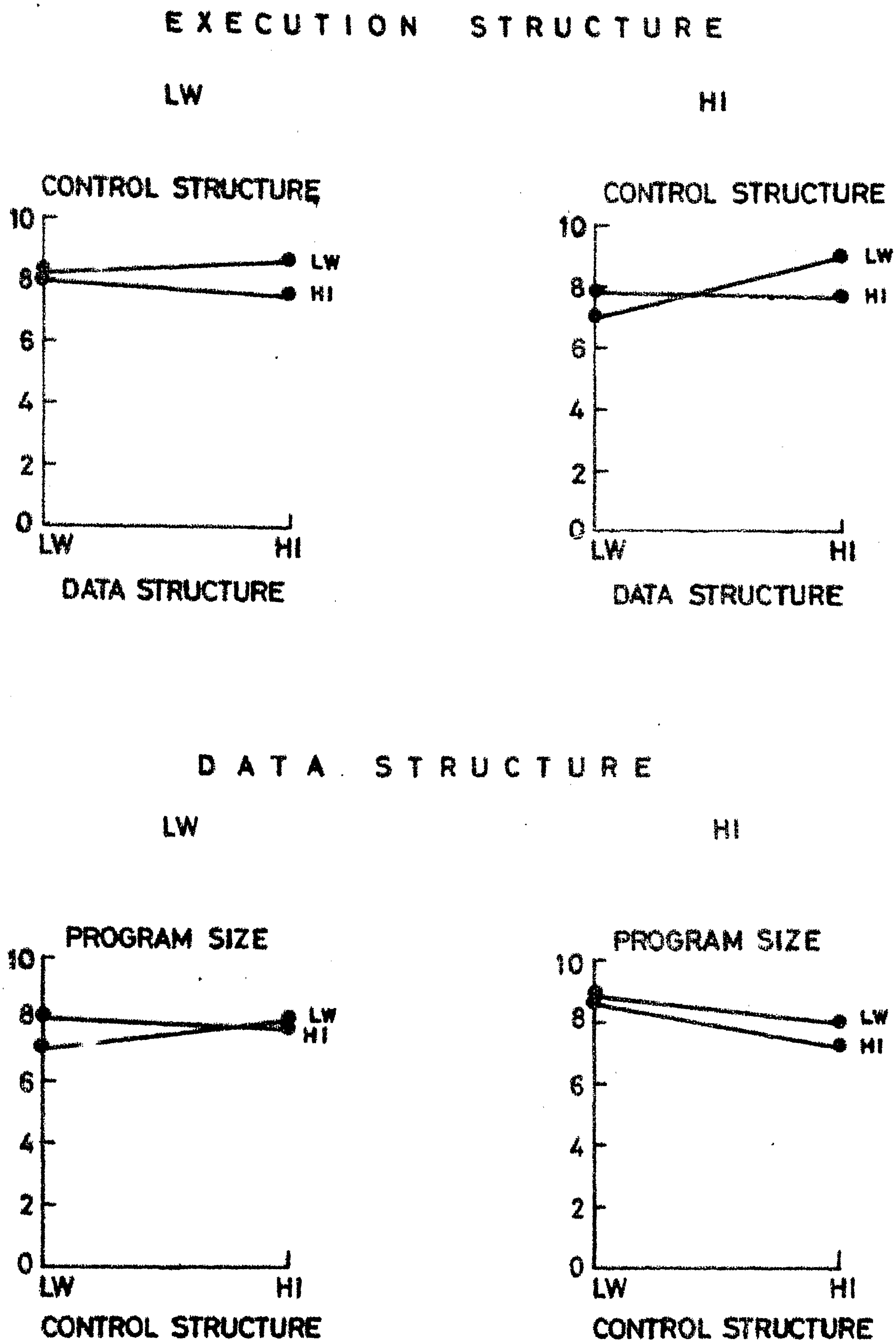
Y-AXIS REPRESENTS MEAN SCORE OF PROGRAM SUMMARY IN ALL CURVES

FIG. 4.8 2-WAY INTERACTIONS IN PROGRAM SUMMARY



Y-AXIS REPRESENTS MEAN SCORE OF PROGRAM SUMMARY IN ALL CURVES

FIG. 4.9 3-WAY INTERACTIONS IN PROGRAM SUMMARY



Y-AXIS REPRESENTS MEAN SCORE OF PROGRAM SUMMARY IN ALL CURVES

FIG. 4.10 3-WAY INTERACTIONS IN PROGRAM SUMMARY

A comparison of curves in Figures 4.1 and 4.8 suggests that the differences in the curves of two figures are quantitative only. There is no qualitative difference. This quantitative difference in the performance of the two groups of the subjects is to be expected. The subjects of the second experiment had a more formal advanced education in program development, methodologies, language features, etc., than those of Experiment 1. They are also expected to be more talented than the other group of the subjects. It is not surprising, therefore, to find that subjects of Experiment 2 did not find the programs as difficult as the first group.

4.3 CONCLUSIONS FROM RESULTS OF EXPERIMENTS

A global view on the results of the experiments described in Chapters 3 and 4 suggest some important conclusions.

The meaningfulness of the expressions, data names and their organization in context of some problem domain has a very dominant effect on program understanding. This conclusion is also supported by the fact that four of the programs used in the experiments described in this chapter, were more difficult to understand due to ambiguity in referential meaning of the variable name, unfamiliarity with the problem domain and the algorithm. The selection of data names, their organization and the organization of the events in program have very important bearing on program understanding.

The execution structure complexity affects the program understanding. This complexity can effectively be controlled by number of assignments in the program and the number of operands in the assignment statement.

The experiments have brought out the significance of data structures in program complexity. Data structure was significant in all the experiments described in this chapter. However, the results of the experiment 2 of Chapter 3 indicated the ineffectiveness of the numerosity of data structures. This contradiction is probably due to the nature of the program selected for that experiment. Programs B and C of that experiment were same as program number 2 of the experiment of this chapter. Since this program had highest complexity rating, the lowest score for that program in the experiments of Chapter 3 becomes expected. So, in general, it can be concluded that the data structure of the program affects program understanding.

The subjects of Experiment 2 of this chapter had identified data structures of the programs as the second major factor contributing to program complexity. They identified control structure complexity as a factor contributing most to the program complexity.

The next chapter discusses the formal abstraction of data structures.

CHAPTER 5

ABSTRACTION OF DATA STRUCTURES

Various methodologies have been suggested to control the complexity due to control structure, data structure, and execution structure. The concept of subroutine helped in reducing the functional complexity of programs. It is well suited to abstract operations on data. Dijkstra, [Dijkstra, 72], and Hoare, [Hoare, 72], advocated the use of only a few control and data constructs which are conceptually simple and easier to abstract formally, to reduce respectively the control and data structure complexities. The impact of these methodological suggestions on program understanding has not been investigated rigorously. Only in the case of control structure, some studies have been made, [Sime, 73], [Sheppard, 78].

The results of the experiments, described in previous chapters, indicated the importance of data structures in program complexity. The selection of data names and organizations which are "natural" in context of some problem domain has profound affect on program understanding. However, the data structure complexity does not increase with increasing the number of vectors and matrics in the program. The complexity metric for ~~fof~~ this factor should be based on the method of abstraction of data structures for mental representation.

It was our earnest desire to investigate the impacts of methodological suggestions, of structuring and abstracting data, on program understanding. But due to unavailability of the subjects familiar with the above concepts, the experiments could not be conducted. FORTRAN has very limited data constructs. This, further, constrained the experimentation on data structures. We decided to pursue the study in formal aspects of the specification technique of data abstraction.

The organisation of the chapter is as follows. The first section briefly describes the concept of data abstraction and different techniques of specifying it. The second section contains a formal definition of completeness and consistency of specifications and a proof of their undecidability. The third section includes an informal comparison of the techniques and suggested modification in one of the technique.

5.1 TYPES OF SPECIFICATION

A software system is collection of data and a set of operations on them. Its specification is a precise statement of the requirements that a software system must satisfy. Specification techniques may be classified in two groups: operational abstraction and data abstraction.

In operational abstraction, the semantics of the operations are defined by their effects on the state of

the data structures. The states of the data structures are abstracted in terms of predicates and axioms. The predicates which define the domain and range of the operations are known as pre- and post-conditions respectively. In other words, the pre-conditions describe a state of data structures in which the operations may be started. The post-condition describes the state of the data structures after the operation terminate. Specification techniques of this group are based on the pioneering works of Floyd, [Floyd, 67] ,Hoare, [Hoare, 69], Dijkstra, [Dijkstra, 75],Manna,[Manna, 74,78], and many others.

In data abstraction, the software system is viewed as collection of operations acting on an object or a class of object with constraint that the state of the object can be known only through invocation of the operations. Operations are defined in terms of relations with other operations. The data structures of the program are not referred in the description. This specification is a 'black box' description and defines only behavioral characteristics of software systems.

The starting point for most of the work on data abstraction has been SIMULA 67 [Dahl, 70]. Many programming languages, e.g. [Dahl, 70],[Liskov, 77]and [Wulf, 74], offer a mechanism for binding together the operations and the storage structure representing a data type. But they do not

provide a representation-independent means for specifying the semantics of the operations. Major contribution in the development of specification techniques of this class are from Parnas, [Parnas, 72, 76, 77], Guttag, [Guttag, 75,76,78] and Liskov et al.,[Liskov, 75].

5.2 SPECIFICATION TECHNIQUES FOR ABSTRACT DATA

A brief introduction of specification techniques advocated by Parnas, [Parnas, 72, 76, 77] and Guttag, [Guttag, 75,76,78], is given below.

In these specification techniques, a software system is viewed as collection of operations acting on an object. The set of operations is partitioned in two disjoint sets. The operations contained in one of the set merely report on state of the object without changing it. These operations are named as V-functions and return values that give information about the data objects, [Parnas, 72]. The operations contained in another set, named as O-functions, change the state of the object without reporting.

In his earlier technique, [Parnas, 72], Parnas defined each operation in terms of 'parameters' (par.), 'possible values' (pv.), 'initial values' (iv.), 'applicability conditions' (ac.), and 'effects' (eff.). 'par.' and 'pv.' give domain and range of the operation. 'ac.' lists the conditions which are to be satisfied for successful

invocation of that operation. 'eff.' defines the relation with other operations. 'iv.' is meaningful only for V-functions and gives the initial values to be reported. This notation served well for examples in which the execution of an O-function were immediately visible and could be described in terms of old and new values of V-functions. However, in the cases where the effects of the call of an O-function are not immediately visible in terms of V-function values until some other O-functions have been executed, the notation was not rich enough to define such O-function. Parnas used natural language to define such effects, named as 'delayed effects', in his earlier examples [Parnas, 72]. Price introduced 'hidden function' to specify the delayed effects in his other examples [Price,73]. The hidden functions are not available to the outside user, i.e., they can not be called. Their purpose is purely descriptive. The effects of some O-functions may be defined in terms of values of hidden functions.

These hidden functions provide information about the data structure that stores the state of the object. They are suggestive of possible implementations. Since the hidden functions give information which is not a requirement, their inclusion in the specification goes against the basic motivation of representation-independent specification. The first step towards writing a specification without hidden functions is finding a set of call sequences such that a description of observable effects of each sequence in the set allow one

to deduce the observable effects of any sequence of calls. Parnas, [Parnas, 77], named such a set as a <u>characterizing</u> set of function call sequences. However, finding this characterizing set for a system is not an easy task.

Bartussek and Parnas, [Bartussek, 78] , proposed a new technique. The new technique allows the specification of modules with delayed effects without any reference to internal data structures.

In the new technique, a specification consists of two parts. The first part, called syntax, gives the names of all the operations and the type of each parameter. The notation to specify the domain and the range of an operation has been adopted from [Guttag, 75]. The second part, called <u>semantics</u>, consists of three types of assertions.

The first set of assertions defines a <u>characterizing set</u> of function call sequences. These function call sequences are named as <u>Legal</u> <u>traces</u> in the new technique. The Legal traces are set of traces such that calling the function as described in the trace (starting with a module in its initial state) will not result in traps.

The second set of assertions define an equivalence relation on traces. The equivalent traces have the same legality (either both are legal or both are not legal) and the same externally visible effects on the V-functions. These assertions extend the class of legal traces. Any

trace that cannot be proved legal by above two sets of assertions are considered illegal.

The last set of assertions are about the values returned by V-functions at the end of traces. These assertions describe the the values returned by V-functions for a subclass of the class of legal traces. The traces used in this last set of assertions are known as <u>normal form traces</u>. Using the equivalence statements, one can derive the values of V-functions at the end of other traces by finding an equivalent normal form trace.

The details of the notation for syntax and semantics of the specification are discussed in [Bartussek, 78]. An example taken from [Bartussek, 78] is given here.

Example: A stack for integer values.

<u>Syntax</u>:

PUSH:<Integer>x<Stack>→<Stack>

POP:<Stack>→<Stack>

TOP:<Stack>→<Integer>

DEPTH:<Stack>→<Integer>

<u>Semantics</u>:

A. Legality:

(1) $\Sigma(T) \Rightarrow \Sigma(T.PUSH(a))$

(2) $\Sigma(T.TOP) = \Sigma(T.POP)$

B. Equivalences:

(3) $\Sigma(T) \Rightarrow T.DEPTH \equiv T$

(4) $\Sigma(T) \Rightarrow T.PUSH(a).POP \equiv T$

(5) $\Sigma(T.TOP) \Rightarrow T.TOP \equiv T$

C. Values:

(6) $\Sigma(T) \Rightarrow V(T.PUSH(a).TOP)=a$

(7) For all integers a, $\Sigma(T) \Rightarrow$

$V(T.PUSH(a).DEPTH)=1+V(T.DEPTH)$

(8) $V(DEPTH)=0$.

In the above specification, $\Sigma(T)$ is a predicate and T is a trace. $\Sigma(T)$ is true if T is a legal trace. Further, if T is legal trace, X is syntactically correct call on a V-function and $\Sigma(T.X)$ then $V(T.X)$ describes the value delivered by X when called after an execution of T. '$\equiv$' is an equivalence relation. Two traces T_1 and T_2 are equivalent if for any subtrace, S, $\Sigma(T_1.S)=\Sigma(T_2.S)$ and $\Sigma(T_1.S.X) \Rightarrow V(T_1.S.X) = V(T_2.S.X)$ where X is a V-function.

The above specification is of a stack with unlimited capacity. The first assertion of semantic part states that if $\Sigma(T)$ is true then $\Sigma(T.PUSH(a))$ will be also true. That means if T is a legal trace then T.PUSH(a) is also a legal trace. So any sequence of PUSH operation is a legal trace. The second assertion states that if T.TOP is a legal trace then T.POP is also a legal trace. That is if TOP can be called after a sequence T then POP can also be called after T. The equivalence section allows one to reduce any legal trace with PUSH, POP, and TOP to one that is equivalent but contains only PUSH operation. The last section, values, defines initial value of DEPTH and values of TOP and DEPTH after PUSH operation.

The formal basis of specification technique advocated by Zilles, [Zilles, 75], and Guttag, [Guttag, 75], stems from the hetrogeneous algebra of Birkhoff and Lipson, [Birkhoff,70]. In these techniques, **an** abstract data type is defined as hetrogeneous algebra T =[TOI+I, F], [Guttag, 75], where TOI ("Type of Interest") is the set of all values of the abstract data types being defined, I is the set of all other defined types and F is the set of operations such that:

a) The range of each $f \in F$ is contained in TOI+I.

b) The domain of each $f \in F$ is a cross product of members of TOI+I and

c) For each $f \in F$, T.OI is either contained in the domain of f, is the range of f, or both.

Like Parnas' technique, the set of operations, F, is partitioned into two disjoint sets. V- and O-function of Parnas are named as O- and S-function respectively. The specification consists of two parts: Syntax and Semantics. The notation for the syntax is same as of the previous technique. Semantic part consists of relations which define the meaning of the operation by stating their relationship to one another. The language for the semantic part is like a programming language. The following features of the programming language have been chosen:

a) free variables,

b) if - then - else expressions

c) Boolean expressions,

d) recursion.

A simple example from [Guttag, 76] is reproduced here to describe the notation for semantic part.

Example: <u>Type</u> Stack [Item]

<u>begin</u>

1. Declare

   NEWSTACK( ) → Stack

   PUSH(Stack, Item) → Stack

   POP(Stack) → Stack

   TOP(Stack) → Item ∪ { Undefined }

   ISNEWSTACK(Stack) → Boolean

2. For all s ∈ Stack, i ∈ Item <u>Let</u>

   ISNEWSTACK(NEWSTACK)=True

   ISNEWSTACK(PUSH(s,i))=False

   POP(NEWSTACK)=NEWSTACK

   POP(PUSH(s,i))=s

   TOP(NEWSTACK)=Undefined

   TOP(PUSH(s,i))=i

<u>end</u>.

In the above specification, the equations within <u>for all</u> and <u>end</u> are the axioms which describe the semantics of the operations. 'Undefined' is treated as additional value.

5.3 COMPLETENESS AND CONSISTENCY OF SPECIFICATION

A problem associated with these techniques, as pointed out by Parnas, [Parnas, 77], and discussed in detail by Guttag, [Guttag, 75], is of knowing when one has completely specified the abstract data type. Guttag has shown for his technique that the problem of establishing whether or not a set of axioms is complete is, in general, undecidable. Guttag has also suggested a general out line of attack to create the axioms so that they are complete [Guttag, 75, 76] .

It is expected that the undecidability of completeness and consistency of the specification will be true also in the case of Parnas technique. In this section, a formal definition of completeness and consistency is given along with the proof of their undecidability. The notation adopted for the proof is based on Parnas earlier (1972) notation. The reason for choosing this old notation instead of recent one, [Bartussek,78] is that we believe the earlier technique with suggested modifications to be superior compared with other techniques in some of the important aspects.

A software module is viewed as collection of operations. The module can be used only through invocation of these operations. The specification defines the module by defining the set of all sequence of call of operation which can be carried out and effect of these call sequences on state of module, starting with initial state of the module. In other words, specification defines the feasible and infeasible

sequences of function calls, starting from initial state of the module, and their effects on the state of the module.

The state of the module or abstract object is characterized by the values stored in the module. These values can be known only through calls on V-functions. The initial values of V-functions define the initial state of the module. The call of a V-function merely returns a value. It does not change the state of the module. The call of O-functions change the state of the module. This change of state is reflected by change in the values returned by call on V-functions after the invocation of O-functions. If for all feasible call sequences of O-functions starting from the initial state of the module, the values returned by V-functions are defined and can be derived from the specification, then the specification is <u>complete</u>. Further, if the value derived is unique then the specification is consistent. In other words, the completeness is ensured if states of the module have only defined values.

This definition of completeness and consistency of the specification is equivalent to that of Bartussek and Parnas, [Bartussek, 78]. Since the call of V-function do not change the state of the module, the effect of a sequence of calls of both O- and V-functions on the state of module will be same if calls of V-functions are removed from the sequences. To make it clear, an example is given below from the specification of stack.

PUSH(a).DEPTH.PUSH(b)≡PUSH(a).PUSH(b)

This above equivalence, '≡', is not in the sense of Bartussek and Parnas, [Bartussek, 78]. The meaning of this equivalence is that the state of the module, after successful calls of left- and right- hand sequence of operation, starting from any state of the module, are same.

The trace defined in [Bartussek, 78], includes both O- and V-functions and so their definition of completeness looks slightly different. But with above definition of '≡', it is clear that both definitions are equivalent.

Definition:

A sequence of call of operations g_1, g_2, ..., g_n from the set of O-functions, denoted as $g_1 o g_2 o g_3 \ldots o g_n$, is said to defined (or feasible) if successful invocation of g_1 may be followed by successful invocation of g_2 and so on, starting from the initial state of the module. If for all defined sequences of operations (including null) from the set of O-functions, the values to be returned by call of operations from the set of V-functions are derivable from the specification and if the values derived is always defined, then the specification is complete. Further, if unique values can be deduced then the specification is said to be consistent.

The notation of Parnas earlier (1972) specification technique did not permit the specification of delayed effects and hence the feasible and infeasible sequence of operation calls. However, this limitation does not create any difficulty

in proving the undecidability of the completeness. The modifications in the notation are discussed in the final section of this chapter.

We now prove the undecidability of completeness of specifications.

Theorem:

There does not exist an algorithm for determining the completeness of an arbitrary specification of abstract data.

The proof is given in two stages. In first stage, it is proved that for a given Turing machine, there exist a specification such that the specification is not complete iff the Turing machine halts. This result is used to establish an equivalence between two problems, halting of Turing machine and the completeness of the specification technique.

Proof:

(a) Let us assume that there exists an algorithm for determining the completeness of the specification. That means there exists a Turing machine, T, to determine the completeness and is guranteed to halt.

(b) The theorem is proved by showing the non-existence of T and for that following lemma is proved first.

Lemma:

For a given Turing machine, there exists a specification such that the specification is not complete iff the Turing machine halts.

Proof:

In the proof, the notations of [Hopcroft, 69] are used. Let the given Turing machine be

$$T_m = (K, \Gamma, \delta, q_o, F), \text{ where}$$

K is the finite set of states,

Γ is the finite set of tape symbols including the blank 'B',

δ is the next move function, a mapping from $K \times \Gamma$ to $K \times (\Gamma - \{B\}) \times \{L,R\}$ and is defined by the set of quintiples known as rules of T_m. That is

$\delta = \{r_1, r_2, \ldots, r_n\}$ where

$r_i = \langle q_j, \alpha, q_k, \beta, d \rangle$ and $q_j, q_k \varepsilon K, \alpha \varepsilon \Gamma, \beta \varepsilon (\Gamma - \{B\})$, and $d \varepsilon \{L,R\}$.

$q_o \varepsilon K$ is the start state,

$F \subseteq K$ is the set of final states.

The specification corresponding to T_m is given by defining the operations from the set of V- and O-functions, named as V-Func. and O-Func. The set V-Func. is defined as

$$\text{V-Func.} = \{v_1, v_2, v_3, v_4\} \text{ where}$$

v_1 returns the present state of T_m as value,

v_2 simulates the tape of T_m and returns the tape symbol under the head of T_m as value,

v_3 has only one defined value, '1', and

v_4 gives the position of the head.

Each operation from V-Func. is defined below in the notation of Parnas, [Parnas, 72].

V-Func.: v_1
par. : none
pv. : $k \in K$
iv. : q_o
ac. : none
eff. : none

V-Func. : v_2
par. : none
pv. : $s \in \Gamma$
iv. : B
ac. : none
eff. : none

V-Func.: v_3
par. : none
pv. : $\{1\}$
iv : 1
ac. : none
eff. : none

V-Func. : v_4
par. : none
pv. : $i \in$ set of integers
iv. : 1
ac. : none
eff. : none

The set O-Func. is defined as

$$\text{O-Func.} = \delta \cup \{r_{n+1}\}.$$

The definition of operations from O-Func. is given below.

O-Func. : $r_i \in \delta$
par. : none
pv. : none
iv. : none
ac. : $v_1 = q_j$ and $v_2 = \alpha$
eff. : $v_1 = q_k$; $v_2 = \beta$
$v_4 = {}'v_4{}' + 1$ if $d = R$ or
$v_4 = {}'v_4{}' - 1$ if $d = L$

O-Func. : r_{n+1}
par. : none
pv. : none
iv. : none
ac. : $v_1 \in F$
eff. : v_3 = 'Undefined'

The semantics of symbol "=" and " ' ' " are same as in [Parnas, 72]. The 'Undefined' is treated here as special value.

The way specification has been constructed ensures that if T_m halts then the specification is not complete. The operation r_{n+1} can be successfully invoked only when T_m has reached in one of the final states and once it is invoked, v_3 of V-Func. become undefined.

(c) If there exists T, T may find whether a specification is complete. In particular, it may find if the specification, built as above, of an arbitrary Turing machine T_m is incomplete. Thus it may determine (lemma) whether T_m halts. Since it is known that there is no Turing machine which will determine if an arbitrary Turing machine will eventually halt, so the assumption (a) must be wrong. And, hence, the theorem is proved.

The proof of the lemma gives some additional information about power of the specification technique. Since there exists a specification for every Turing machine, it implies that technique is powerful enough to specify any computable function. Selection of suitable V-Functions has avoided the 'delayed effects' in the specification.

5.4 MODIFICATION AND COMPARISON OF PARNAS' TECHNIQUE WITH OTHER TECHNIQUES

The three techniques considered for comparisons are [Parnas, 72], [Guttag, 75], and [Bartussek, 78], and they

are referred as technique number 1, 2, and 3 respectively in this section.

All the three techniques are powerful enough to specify any computable function. The completeness and consistency of the specifications are undecidable in all the three techniques. The power, completeness and consistency problem of the technique 2 is discussed in [Guttag, 75].

The major difference between the techniques are due to different schema used for defining the semantics. The notation of technique 1 does not permit the specification of 'delayed effects'. The notation of this technique is not rich enough to characterize all feasible and infeasible call sequences. Hidden functions have to be used for specifying the delayed effects. We suggest some modifications in this technique to enhance its notation.

5.4.1 Modifications:

The first modification is suggested to facilitate the specification of delayed effects. The notation of technique 1 does not permit the specifications of infeasible sequence of operation and of any special property associated with the sequence of operations.

As pointed earlier, only the sequence of call of operation from the set of O-functions are important because they only change the state of the module. Some of the properties of the sequence of call of O-function may not

be defined with the definition of individual operation of the sequence. Such properties may be specified in terms of values of V-function before and after the invocation of the sequence. The suggested modification in the notation permit the specification of such sequences.

The notation is given below.

< Relations > ::= [< Sequence of operation calls from O-functions >
< effects >] *

'Relations' signifies the special property associated with a sequence. It is defined by naming the sequence and specifying its effect in terms of values of V-functions. The values returned by affected V-function may have special value 'ERROR' to indicate that this sequence is not permissible. The execution of such sequence of operation would result in error. An example of the notation below is given by defining the delayed effect of POP operation on TOP of stack [Parnas, 72].

Relation : PUSH(a).POP
eff. : TOP='TOP'
DEPTH='DEPTH'

The above specification states that if PUSH(a) operation is followed by POP, the value returned by TOP and DEPTH after the execution of this sequence will remain same as before the execution of the sequence. The fourth predicate of the

first example of the first section defines this property. Similarly, boundedness of the stack can be defined under this notation.

The second suggested modification is about specification of exception handling strategy. In technique 1, the operations can be called if and only if the condition listed in 'ac.' are satisfied. The module will trap in states where conditions are not satisfied. There is no record of previous invocation of operation if error has occurred. It is the responsibility of the invoker or user of the module to respond to the condition of error. The imposition of this constraint may be helpful in reducing some burden of the implementor from the interfacie problems. But it will be preferable to have a feature in the specification technique by which one can define any general exception handling strategy. This aspect has been ignored in both technique 2 and 3 also.

In modified notation, 'ac.' will contain not only the conditions to be met for successful invocation of the operation but also have information about the action to be taken if the conditions in 'ac.' are not satisfied. The action may include passing the information about the source of error, a default action or a resume option to the invoker. The proposed notation is given below.

```
<ac >::= [<Condition> |<ac > [<Code=integer>]*
          |<ac >[<Signal=Boolean Values>]*
          |<ac >[<Default=action name>]* ]*
```

[ ] * indicates repitition.

If 'ac.' contains only conditions then the operation is terminated if these conditions are not satisfied. No history of call is kept in the system and no action is taken for the exception. If in 'ac.' conditions are followed by 'codes', the operation is terminated and an integer code, identified with the condition is passed to the invoker. So in this case user knows the source of error. In the case of signal, the operation will be either terminated or resumed under the direction of the user. If some default action is to be taken that also may be listed. The above notation is based on the work of Goodenough, [Goodenough, 75].

We expect that the inclusion of these modification will increase the range of applicability of the technique.

5.4.2 Comparison:

In technique 2, the semantics is defined in the form of set of identities of axioms. Both V- and O-functions are treated exactly in the same way for describing their definition This is in contrast with technique 1, where only effected operations of V-functions are included in the 'eff.' section.

The notation for semantics, in technique 2, does not limit the depth of nesting on both side of identities. However, for constructing a specification which is complete, the depth of nesting on left-hand side of identities is limited to two only [Guttag, 75]. This limitation on depth of nesting has necessitated the use of hidden functions. Guttag, [Guttag, 76], have to use hidden function DEPTH. Further, limiting the depth of nesting on left-hand side of identities may increase the size of the axiom.

The notation for the semantic part in technique 3 is better in many respect as compared to technique 2. Since the technique 3 provides for the definition of all legal sequences of function calls, the need for hidden functions are removed. There is no limitation, from the notation, on the length of the sequence of function calls, the characterization of legal and illegal sequences.

It is easier to prove the correctness of the implementation of data abstracts in technique 2 than in other techniques. The duality between axioms and programs has been very fruitfully exploited to prove the implementation [Guttag, 78] . The language used for representing programs and axioms are same. Due to this commonality of language, one can view axioms as programs and vice-versa. This duality also gives an implementation-direct implementation. This

implementation is in terms of tree structures generated by V-functions. The direct implementation can also serve as an aid in constructing specifications.

In other two techniques, to prove the correctness of implementation, language other than specification language has been used. The operations and their implementations i.e. programs are defined as predicate transformer. The equivalence of two predicate transformer is proved with the help of homomorphic mapping from abstract data to implementation [Bartussek, 76]. The proof becomes difficult due to use of two different languages for specification and implementation.

The technique 1 has some special features which have been ignored in other techniques. The initial values of V-functions are defined explicitly. In other two techniques, they are defined through axioms and assertions. In our opinion, explicit definition is better than the implicit.

The O-functions are defined by giving their effect on set of V-functions. The 'eff' section includes only affected V-functions. In situations, where each O-function affects only a few V-functions, this notation will result in small size of the specification. With the proposed modification, the technique 1 may prove better from readability and understandibility point of view. However, the technique 2 is better in regard of proving correctness of implementation.

CHAPTER 6

CONCLUSIONS AND SUGGESTIONS

6.1 CONCLUSIONS

We have demonstrated that four algorithm-dependent features: size, control structure, data structure, and computation structure of the program contribute independently to perceived program complexity. The above four factors of program complexity affect program understanding. Simple syntactic measures of the factors, such as counting number of IF statements [Gilb, 77], vectors and matrices, do not reflect their contribution in psychological complexity of programs. In our opinion, the other measures of control complexity are also rather superficial. All these measures reflect the numerosity of certain features of control structures such as number of basic paths [McCabe, 76], number of knots [Woodward, 79], etc. They do not take into account regularity or structure in different features of a factor. Since human beings are very efficient in identifying regularity or structure at various levels, any regularity in the features of a factor aid understanding. The measure of complexity of a factor should take into account both numerosity and regularity of different features of the factor.

Program features which establish an association or suggest an analogy with a problem domain, termed collectively as factor of 'meaningfulness', have an important bearing on program understanding. It is evident from the experiments that uncontrolled factors in the program contributed more to program complexity than the controlled ones. We believe that subjective rating may prove to be a good measure of program-complexity provided a serious exercise with programs such as writing a program description or program modification, etc., is carried out to aid understanding. The exercise will make the rating more dependable.

We have shown that Parnas' early specification technique (1972) is powerful enough to specify any computable function; further, the completeness and consistency of a specification are undecidable. We have suggested some modifications in the above technique to specify exception handling strategy and 'delayed effects', [Parnas, 72]. We believe that the modified technique has some good features which would make it more useful than other techniques ([Guttag, 75], [Bartussek, 78]).

6.2 LIMITATIONS

Programs used in the experiments were selected from various programming text-books and were modified to meet the criteria of selection. The controlled variation of factors of interest and elimination of uncontrolled effects as far

as possible made the search of real life programs for the experiments very difficult. This led us to use text-book programs for the experiment. Time-constraint of the subjects did not permit the selection of large programs. The above two factors may cast some doubt on generalizability of these results to real life programs. However, we believe that the above factors do not limit the applicability of these results in real life situations. Most of programming methodologies emphasize dividing the problem in smaller parts to control the complexity. At program level, these small parts are subroutines, procedures or modules. The typical size of subroutines or modules may range between 20 to 100 lines. The overall complexity of the program will depend on the complexity of constituent parts or modules, and their interactions. We envisage the application of the results of the thesis at module levels. For interaction among modules and their contribution to overall program complexity, further experimentations are required.

FORTRAN, the language used in the experiments, has limited data structuring constructs such as vectors and matrices. It does not have constructs for records, pointers and other interesting data structures. This limitation prohibited us from experimenting with various important aspects of data structuring and their effect on program comprehension.

In some of the experiments, novice programmers served as subjects. We did not consider experience as a factor in evaluation of expression. The task of expression evaluation involves following a mechanical procedure and the effort needed for this task is expected to show a marginal change with the experience Similarly, for Experiment 2 of Chapter 3, we expected only quantitative difference in results with experienced programmers. In our opinion, the effect of experience depends on the nature of task. The performance of novice and experienced programmers did not show any difference in one of the experiment conducted by [Shneiderman 77].

6.3 OBSERVATIONS

Some interesting aspects of program understanding and of experimentation in them were revealed during informal talk with subjects after experiment, from comments written during experiment and by observing the subjects during the period they read and understood the program through their rough work.

In Experiment 3 of Chapter 3, the subjects were required to evaluate the values of variables at different points of the program. It was expected that this exercise would aid in understanding. Twelve subjects wrote that they could not reconstruct the complete program because they studied only that section or path of the program which was to be executed

for evaluation of values of variables asked for. In case of some other subjects, it was evident from their reconstructed programs that they also understand only a section of programs which was necessary for evaluation of values of variables.

These comments and observation suggest that evaluation of values did aid in understanding. However, the values of variables could be evaluated even by partial understanding of the program. If the evaluation of values of variables at different points of the program is to be used as a measure of program understanding, then the care should be taken in selection of variables and points in the program for their evaluation. These variables and points of evaluation of their values should account for all possible paths of the program. Various algorithms reported in the literature for test data selection may be applied to decide about variables and the points at which their values are to be evaluated. Without such precaution, the performance of the subjects in evaluation of the values may not reflect their understanding of program.

In the experiments (Experiments 1 and 2 of Chapter 4), subjects used the blank portion of the program sheet as scratch-pad for some calculation, drawing flowcharts, comments, etc. These rough works were indicative of their activities during program reading and understanding and revealed partially the different methods of abstracting

information about the program. One of the subjects always drew the complete flowchart of the program he was studying. Two other subjects used to identify the section of the program with some problem domain dependent events by tracing the control blocks of the program. For example, the subjects had identified a block by drawing aline from start of DO loop to its end and wrote 'initialization of variables' near the line. The complete program was divided in many such blocks. One subject always started making tables correspon- ding to data structure of the program. Table had as many entries as the number of simple variables read through I/O statement. Entries consisted of variable name and their format. Similarly, for vectors and matrices he would draw a separate line structure. He identified various events of the program with changes in the above structure. The scratch-pad work mostly contained the line structures, variables names and their values at different stages of the program and comments. Such preliminary exercises of the rest of the subjects did not exhibit any pattern. Their scratch-pad contained only few calculations, some variable names,etc.

The experiments were not designed for investigation of various methods of abstraction in program understanding. The scratch-pads of the subjects were also not very infor- mative. It will be premature to conclude about methods of abstraction from above observations.

In Experiment 1 of Chapter 4, subjects indicated in their comments and through informal talk after experiment that some of the programs were more difficult. The source of increased complexity, as identified by the subject, were not due to controlled factors of the experiment. In one program (No.2) the difficulty was created by one variable (REQ). Many of the subjects could not make out the purpose of the variable and what it represented. In another program (No.6), the complexity was due to switching of a variable, JW. Similarly, in some other programs (No. 10, 16) the complexities were due to unfamiliarity of the subjects with algorithm and formulae used in the programs. These various uncontrolled factors had a very significant effect on program understanding.

The above observation is a pointer to the need of elimination of uncontrolled factors as far as possible for the success of experiments.

6.4 DIRECTION FOR FURTHER WORK

The observations of the previous section are suggestive of the direction of the future work. As we have pointed out earlier, the various metrics of complexity of program factors do not reflect their psychological complexity. There are no measures for complexity of data structure and computation structure of programs. Efforts should be directed towards defining measures for various factors of program complexity

based on validated theory of program comprehension. Experiments should be designed to study various methods of abstraction employed by human beings in program understanding. Formation of mental representation of programs, strategies adopted for learning the program and the impact of various methodological suggestions on these processes should be thoroughly investigated.

Various measures of program comprehension such as hand simulation of programs, responding to questionnaires about the program, memorization of the program, etc. have been used in experiments reported in the literature. Among all such measures, program memorization has been used most widely and accepted as a good measure. However, no comparative evaluation of different measures has been done. The superiority of a measure should be established through experiments.

Results of the experiments in program comprehension are of qualitative nature. Qualitative results do help in identifying the problem but they do not provide any quantitative basis for enginineering design and standardisation. For example, the quantification of optimal size of subroutine, the depth of nesting of DO loops, number of Boolean variables in a conditional statement, etc., from program understanding point of view, will give a sound basis for good engineering designs.

Finally, effort should be directed towards standardising the development of programs for the experimentation to eliminate uncontrolled factors as far as possible. Various syntactic transformations may be employed to get different versions of programs for experiments.

REFERENCES

[Amster, 76] Amster, S.J, Davis, E.J., Dickman, B.N., and Kuoni, J.F., "An experiment in automatic quality evaluation of software", Proc. Symp. Computer Software Engg., MRI Symposia Series, Vol.XXIV, Fox, J., (Ed.), Polytechnic Institute of New York, 1976, pp. 171-197.

[Anderson, 65] Anderson, N.H., "Averaging versus adding as stimulus combination", Journal of Experimental Psychology, Vol.70, No.4, 1965, pp. 394-400.

[Anderson, 72] Anderson, N.H., "Information Integration Theory: A brief survey", in Contemporary Developments in Mathematical Psychology, Krantz, D.H., Atkinson, R.C., Luce, R.D., and Suppes, P., (Eds.), Vol.2, Freeman, San Francisco, 1974.

[Anderson, 74a] Anderson, N.H., "Methods for studying information integration", Technical Report, No. CHIP-43, University of California, San Diego, June 1974.

[Anderson, 74b] Anderson, N.H., "Algebraic models for information integration", Technical Report, No. CHIP-45, University of California, San Diego, June 1974.

[Anderson, 76] Anderson, N.H., "How functional measurement can yield validated interval scales of mental quantities", Journal of Applied Psychology, Vol.61, No.6, 1976, pp. 677-692.

[Anderson, 77] Anderson, N.H., "Note on functional measurement and data analysis", Perception and Psychophysics, Vol.21, No.3, 1977, pp. 201-215.

[Bartussek, 76] Bartussek, W., and Würges, H., "Proving that an implementation meets its abstract specification", Forschungsbericht BSI 76/2, Technische Hochschule Darmstadt, Fedral Republic of Germany, May 1976.

[Bartussek, 78] Bartussek, W., and Parnas, D.L., "Using traces to write abstract specifications for software modules", Preliminary Draft of Technical Report, Dept. of Computer Science, University of North Carolina at Chappel Hill, Chappel Hill, 1978.

[Birkhoff, 70] Birkhoff, G., and Lipson, J.D., "Hetrogeneous Algebras", Journal of Combinatorial Theory, Vol.8, 1970, pp. 115-133.

[Bobrow, 75a] Bobrow, D.G., "Dimensions of representation", in Representation and Understanding; Studies in Cognitive Science, Bobrow, D.G., and Collins, A., (Eds.), Academic Press, Inc., New York, 1975, pp. 1-34.

[Bobrow, 75b] Bobrow, R.J., and Brown, J.S., "Systematic Understanding: Synthesis, analysis, and contingent knowledge in specialised understanding systems", in Representation and Understanding; Studies in Cognitive Science, Bobrow, D.G., and Collins, A., (Eds.), Academic Press, Inc., New York, 1975, pp. 103-131.

[Brooks, 77] Brooks, R., "Towards theory of cognitive processes in computer programming", Int. Journal of Man-Machine Studies, Vol.9, 1977, pp. 737-751.

[Dahl, 70] Dahl, O.J., Myhrhang, B., and Nygaard, K., "The SIMULA 67 common base language", Publication S-22, Norwegian Computing Centre, Oslo, 1970.

[Dalal, 78] Dalal, A.K., "Expected job attractiveness and satisfaction as information integration", Ph.D. dissertation, Indian Institute of Technology, Kanpur, June 1978.

[Dijkstra, 72] Dijkstra, E.W., "Notes on structured programming", in Structured Programming, Dahl, O.J., Dijkstra, E.W., and Hoare, C.A.R., Academic Press, 1972.

[Dijkstra, 75] Dijkstra, E.W., "Guarded commands, nondeterminanc and formal derivation of programs", Commn. ACM., Vol.18, No. 8, Aug. 1975, pp. 453-457.

[Elshoff, 76a] Elshoff, J.L., "An analysis of some commercial PL/1 programs", IEEE Transactions on Software Engineering, Vol. SE-2, No.2, 1976, pp.113-120.

[Elshoff, 76b] Elshoff, J.L., "A numerical profile of commercial PL/1 programs", Software-Practice and Experience, Vol.6, 1976, pp.505-525.

[Floyd, 67] Floyd, J.W., "Assigning meaning to programs", Proc. Symposium in Applied Mathematics, Vol.XIX, American Mathematical Society, 1967, pp.19-32.

[Gilb, 77] Gilb, T., Software Metrics, Winthrop Publisher Inc., Cambridge, Massachusetts, 1977.

[Goodenough, 75] Goodenough, J.B., "Exception handling and a proposed notation", Commn. ACM., Vol.18, No.12, Dec.1975, pp. 683-696.

[Gordon, 75] Gorden, R.D., and Halsted,M.H., "An experiment comparing FORTRAN program times with the software physics hypothesis", Technical Report, No.167, Computer Science Department, Purdue University, Layfayette, 1975.

[Guttag, 75] Guttag, J.V., "The specification and application to programming of abstract data types", Ph.D. dissertation, Computer System Research Group, University of Toronto, Sept. 1975.

[Guttag, 76] Guttag, J.V., Horwitz, E., and Musser, D.R., "The design of data type specifications", Research Report, No.ISI/RR-76-49, Information Science Institute, University of Southern California, California, Nov. 1976.

[Guttag, 78] Guttag, J.V., Horwitz, E., and Musser, D.R., "Abstract data types and software validation", Commn. ACM., Vol.21, No.12, Dec.1978, pp.1048-1063.

[Halstead, 75] Halstead, M.H., "Toward a theoretical basis for estimating programming efforts", Technical Report, No. CSD-TR 143, Purdue University, Layfayette, May 1975.

[Hoare, 69] Hoare, C.A.R., "An axiomatic basis for computer programming", Commn. ACM., Vol.12, No.10, Oct.1969, pp.576-580.

[Hoare, 72] Hoare, C.A.R., "Notes on data structuring", in Structured Programming, Dahl, O.J., Dijkstra, E.W., and Hoare, C.A.R., Academic Press, 1972.

[Hopcroft, 69] Hopcroft, J.E., and Ullman, J.D., Formal Language and Their Relation To Automata, Addison-Wesley Pub.Comp., Reading, Massachusetts, 1969.

[Horowitz, 72] Horowitz, L.M., and Manelis, L., "Toward a theory of redintegrative memory: Adjective-noun phrases", in The Psychology of Learning and Motivation, Bower, G.H., (Ed.), Academic Press, New York, 1972.

[Liskov, 75] Liskov, B.H. and Zilles, S.N., "Specification techniques for data abstraction", IEEE Transaction on Software Engineering, Vol.SE-1, No.1, March 1975, pp. 7-18.

[Liskov, 77] Liskov, B., Snyder, A., Atkinson, R., and Schaffert, C., "Abstraction mechanism in CLU", Commn. ACM., Vol.20, No.8, Aug. 1977.

[Löfgren, 77] Löfgren, L., "Complexity of descriptions of systems: a foundational study", Int. Journal of General Systems, Vol.3, 1977, pp.197-214.

[Löfgren, 78] Löfgren, L., "Some foundational views on general systems and the Hempel paradox", Int. Journal of General Systems, Vol.4, 1978, pp. 243-253.

[Manna, 74] Manna, Z., Mathematical Theory of Computation, New York: McGraw-Hill, 1974.

[Manna, 78] Manna, Z., and Waldinger, R., "Is"sometimes" sometimes better than "Always"?", Commn. ACM., Vol.21, No.2, Feb. 1978, pp. 159-172.

[McCabe, 76] McCabe, T.J., "A complexity measure", IEEE Transaction on Software Engineering, Vol. SE-2, No.4, Dec. 1976, pp. 308-320.

[Mills, 72] Mills, H.S., "The complexity of programs", in Program Test Methods, Hetzel, W.C., (Ed.), Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

[Moll, 77] Moll, J.D., and Williges, R.C., "Motion versus pattern cues in visually time compressed target detection in static noise", Journal of Applied Psychology, Vol.77, 1977, pp. 96-103.

[Newell, 75] Newell, A., and Simon, H.A., Human Problem Solving, Prentice-Hall, Englewood Cliff, N.J., 1972.

[Parnas, 72] Parnas, D.L., "A technique for software module specification with examples", Commn. ACM., Vol.15, No.5, May 1972, pp. 330-336.

[Parnas, 76] Parnas, D.L., Handzel, G., and Würges, H., "Design and specification of the minimal subset of an operating system family", IEEE Transaction on Software Engineering, Vol.SE-2, No. 4, Dec. 1976, pp. 301-307.

[Parnas, 77] Parnas, D.L., "The use of precise specifications in the development of software", in Information Processing 77, Gilchrist, B., (Ed.), North-Holland Pub.Comp. 1977.

[Price, 73] Price, W., "Implication of virtual memory family of operating systems", Ph.D. dissertation, Carnegie-Mellon University, June 1973.

[Ronback, 75] Ronback, J.A., "Software reliability: How it effects system reliability", Micro-electronics and Reliability, Vol.14, No.2, April 1975, pp. 121-140.

[Roth, 77] Roth, B. Hayes-, "Evolution of cognitive structure and processes", Psychological Review, Vol.84, No.3, 1977, p. 260-278.

[Schick, 78] Schick, B.J., and Wolverton, R.W., "An analysis of competing software reliability models", IEEE Transaction on Software Engineering, Vol.SE-4, No.2, March 1978, pp. 104-120.

[Shanteau, 69] Shanteau, J.C., and Anderson, N.H., "Test of conflict model for performance judgement", Journal of Mathematical Psychology, Vol.6, 1969, pp.312-325.

[Shanteau, 72] Shanteau, J.C., and Anderson, N.H., "Integration theory applied to judgements of value of information", Journal of Experimental Psychology, Vol.93, 1972, pp.63-68.

[Sheppard, 78] Sheppard, S.B., Borst, M.A., and Love, L.T., "Predicting software comprehensibility", Technical Report, No.TR-77-388100-1, Information System Program, General Electric, Virginia, February 1978.

[Shneiderman, 76] Shneiderman, B., "Exploratory experiments in programmer behaviour", Int. Journal of Computer and Information Science, Vol.5, No.2, 1976, pp.123-143.

[Shneiderman, 77a] Shneiderman, B., Mayer, R., Mckay, D., and Heller, P., "Experimental investigations of the utility of detailed flowcharts in programming", Commn. ACM., Vol.20, No.6, June 1977, pp.373-381.

[Shneiderman, 77b] Shneiderman, B., "Measuring computer program quality and comprehension", Int.Journal of Man-Machine Studies, Vol.9, 1977, pp. 465-478.

[Shneiderman, 79] Shneiderman, B., Software Psychology: Human Factors in Computer and Information Systems, Winthrop Publ.Comp. Cambridge, Massachusetts, 1979, (to appear)

[Sime, 73] Sime, M.E., Green, T.R.G., and Guest, D.J., "Psychological evaluation of two conditional constructions used in computer languages", Int. Journal of Man-Machine Studies, Vol.5, No.1, 1973, pp.105-113.

[Singh, 75] Singh, R., "Information integration theory applied to expected job attractiveness and satisfaction", Journal of Applied Psychology, Vol.60, 1975, pp. 621-623.

[Singh, 79] Singh, R, Gupta, M.,and Dalal,A.K., "Cultural difference in attribution of performance: An integration theoretical analysis", Journal of Personality and Social Psychology, in press.

[Thayer, 75] Thayer, T.A., "Understanding software through empirical reliability analysis", in 1975 Sprint Joint Computer Conference, AFIPS Conf. Proc., Vol.44, AFIPS Press, Montvale, 1975, pp.335-341.

[Weissman, 73] Weissman, L., "Psychological complexity of computer programs: An initial experiment", Technical Report,No.CSRG-26, Computer System Research Group, University of Toronto, July 1973.

[Weissman, 74] Weissman, L., "A methodology for studying the psychological complexity of computer programs", Technical Report, No. CSRG-37, Dept.of Computer Science, University of Toronto, August, 1974.

[Winer, 71] Winer, B.J., Statistical Principles in Experimental Design, McGraw-Hill, New York, 1971.

[Woodward, 79] Woodward, M.R., Hennel, M.A., and Hedley,D., "A measure of control flow complexity in program text", IEEE Transaction on Software Engineering, Vol.SE-5, No.1, Jan.1979, pp.45-50.

[Wulf, 74] Wulf, W.A., "ALPHARD: Toward a language to support structured programming", Technical Report, Dept.of Computer Science, Carnegie-Mellon University, Pittsburg, April 1974.

[Zilles, 75] Zilles, S.N., "Data algebra: A specification technique for data structures", Ph.D. dissertation, Massachusetts Institute of Technology, Project MAC, 1975.

APPENDIX A

STIMULUS COMBINATIONS FOR PRACTICE

END ANCHORS

PRA1

Program size	= Very small
Control structure	= Extremely simple
Data structure	= Extremely simple
Executional complex.	= Low

PRA2

Program size	= Very big
Control structure	= Extremely complex
Data structure	= Extremely complex
Executional complex.	= Very high

PRACTICE EXAMPLES

PRA3

Program size	= Small
Control structure	= Simple
Data structure	= Simple
Executional complex.	= Low

PRA4

Program size	= Small
Control structure	= Simple
Data structure	= Very complex
Executional complex.	= Moderate

PRA5

Program size	= Small
Control structure	= Moderately complex
Data structure	= Very complex
Executional complex.	= Low

PRA6

Program size	= Moderate
Control structure	= Moderately complex
Data structure	= Moderately complex
Executional complex.	= Low

PRA7

Program size	= Moderate
Control structure	= Moderately complex
Data structure	= Moderately complex
Executional complex.	= Moderate

PRA8

Program size	= Big
Control structure	= Simple
Data structure	= Moderately complex
Executional complex.	= Low

PRA9

Program size	= Big
Control structure	= Moderately complex
Data structure	= Moderately complex
Executional complex.	= High

PRAO

Program size	= Big
Control structure	= Very complex
Data structure	= Very complex
Executional complex.	= High

APPENDIX B

JUDGEMENTAL EXPERIMENT

INSTRUCTION

We are engaged in developing a measure for the complexity of programs. This measure can be coupled with a suitable software reliability models for predicting such things as the number of errors remaining in the software package, the expected time to next error, etc. In this experiment, you have to give your judgement about the program complexity on the basis of various information given to you about the program.

The program complexity may be characterized by a combination of four different dimensions of the program. The four dimensions are:

(i) Program size complexity

(ii) Control structure complexity

(iii) Data structure complexity, and

(iv) Execution structure complexity

These complexities can vary continuously. But for convenience, we have divided each complexity in three levels. The program size may be specified in terms of number of different symbols and their frequency of occurrences in the program. The symbols are variable names, operator symbols, punctuation marks, different language constructs, etc. The

program size is divided in levels, small, moderate, and big. One may assume 10 lines program as small and 500 lines program as big

The control structure complexity is due to logical decisions in the program. Each logical decision introduces a separate possible path of computation. The control structure complexity may be defined in terms of number of possible paths in the program and the number of elementary Boolean values used to identify each path. Three levels for this complexity are simple, moderately complex, and very complex. A straight line program or basic block may be an example of simple. The programs having large number of control constructs and associated Boolean values can be considered as very complex.

The data structure characterizes the relation between the data elements. Some example of data structure are array, stack, queue, files, records, etc. The complexity of data structure may be characterized in terms of variable names needed to define the data structure. In the case of array two variable names are needed to form this data structure. The three levels of this complexity are simple, moderately complex, and very complex. One may consider a FORTRAN program operating on only single variables as having a data structure complexity zero or extremely simple. On the other hand, a COBOL program with wide variety of files and records is an example of very complex data structure.

The execution structure complexity emphasizes the severity of each assignment in the program. The executional complexity may be defined in terms of number of operands needed for the execution of the statement and the number of assignments made to variables. The three levels of this complexity are <u>low</u>, <u>moderate</u>, and <u>high</u>.

You consider the information available about all the four dimensions of the complexities of a program and then judge how complex is the program along a 31-point scale. The scale is made up of 31 equally spaced holes on a wooden frame. All holes are identical. The leftmost hole on the scale represents very low program complexity and right-most hole on the scale represents extremely high program complexity. Read information about one program, judge its complexity and insert the wooden peg in the hole which represents the complexity of the program considered.

You will be judging 137 different programs. In some cases you will know only two dimensions of the program; in other cases you will have information about all the four dimensions. You try to base your judgement of the program complexity on the supplied information only. That is, when the information about two dimensions are available, use that information alone; when information about all the four dimensions are supplied, use all of them in making your judgements.

To make you familiar with the nature of the task, use of response scale and your role as judge, we shall give you 10 practice examples. You work with all these examples one by one. During the practice period, feel free to ask questions if you have any. After the practice period, take one card at a time, read the typed information pertaining to the program and then judge its complexity. Finish judgement of all the 137 cards in this manner.

REMEMBER that you are judging PROGRAM COMPLEXITY

Thanking you.

APPENDIX C

EXPRESSION EVALUATION

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
COMPUTER CENTRE

TA-306 Lecture Quiz 3 Feb. 1979
Art of Computation

Given the values of variables:

A	= 16.0	N1	= 5
CENTGR	= 40.0	N2	= 8
H	= 5.0	N3	= 2
R	= 10.0	N4	= 10
U	= 20.0	MN	= 6
T	= 5.0	N5	= 4

Find the values of variables calculated by assignments below in the form of simplified fraction. Numbers produced by calculators are not acceptable. Mark any error found and calculate after correcting.

(i) S=U*T+0.5*A*T*T

S=

(ii) FARNHT=32.0+CENTGR*9.0/5.0

FARNHT=

(iii) MEAN=(N1+N2+N3+N4)/N5

MEAN=

(iv) VØLUM=3.142*R*R*H/3.0

VØLUM=

(v) AVERAG=(N1+N2+N3+N4)/N5

AVERAG=

(vi) XQ=S*A+T*U/(R+H)

SQ=

(vii) MXS=(N1+N2)*(N3+N4)/N5

MXS=

(viii) ZPC=X*T+5*T*MN

ZPC=

(ix) TSX=100.0*A+R+5.0

TSX=

(x) PTS= H+R+T*U/10.0

PTS=

APPENDIX D

DATA STRUCTURE COMPLEXITY

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
COMPUTER CENTRE

TA-306 Lecture Quiz V April 1979

Art of Computation

INSTRUCTIONS

This quiz is meant to test your ability to understand FORTRAN programs. The complete quiz is of one hour duration.

In first part, read carefully the program given to you and try to understand it. You have 25 minutes to do it. Make sure you write your name and roll number on the program sheet, because each one is being given a different program to understand.

After 25 minutes, we will collect the program sheets and distribute answer books. Write a description of the function (what the program does) and algorithm (how it does the work). This task has been allotted 10 minutes. If you did not understand some point about the program or find an error in it please mention that in your description. Question about errors will not be entertained.

When we announce the end of the 10 minutes allotted to description writing, stop that work and write the whole program as you remember it. It is not necessary to get trivial details such as exact variable names, statement numbers, formats, etc. We are only interested in how well you remember the working of the program.

APPENDIX E

PROGRAMS FOR DATA STRUCTURE COMPLEXITY EXPERIMENT

PROGRAM A

```
C       NUMERICAL COMPUTATIONS
        NM=0
   5    READ 10,AB,AX,CG,YP,LZ,CQ,MR,ZS,NT,BS
  10    FORMAT(4F8.2,I1,F4.2,I1,F4.2,I2,F4.2)
        IF(NT.EQ.0) GO TO 50
        DX=AB+CQ
        EN=DX+YP
        PRINT 15,EN
  15    FORMAT(1X,F8.2)
        IF(MR.GT.1) GO TO 25
        GZ=AX+BS
        HC=GZ*AX+10.0
        PRINT 20,HC
  20    FORMAT(1X,F10.2)
        GO TO 40
  25    IF(MR.GT.2) GO TO 35
        BL=CG+ZS
        CQ=CQ+BL
        PRINT 30,CQ
  30    FORMAT(1X,F8.2)
        GO TO 40
  35    IF(MR.GT.3) GO TO 50
        BN=LZ*MR
        CL=CQ+BN
        PRINT 30,CL
  40    NM=NM+1
        GO TO 5
  50    STOP
        END
```

PROGRAM B

```
C  02   CALCULATION OF PERCENTAGE OF SEATS FILLED UP
        INTEGER DT,CE,DO,CO,ST,RQ,MX
        READ 5,KP
    5   FORMAT(A1)
        READ 10,DT,CE,ST,RQ,MX
   10   FORMAT(A4,I2,I2,2I3)
   15   IT=1
        DO=DT
        CO=CE
        NS=1
   20   NT=1
        SC=FLOAT(RQ)/FLOAT(MX)
        READ 10,DT,CE,ST,RQ,MX
        IF(CO.NE.CE) GO TO 25
        NT=NT+1
        IT=IT+1
        SC=SC+FLOAT(RQ)/FLOAT(MX)
        GO TO 20
   25   IF(DO.NE.DT) GO TO 35
        NS=NS+1
        AG=100.0*SC/FLOAT(NT)
        PRINT 30,DO,CO,AG
   30   FORMAT(1X,A4,2X,I2,3X,F8.2)
        CO=CE
        GO TO 20
   35   SG=FLOAT(IT)/FLOAT(NS)
        PRINT 40,DO,SG
   40   FORMAT(1X,A4,2X,F8.2)
        IF(DT.EQ.KP) STOP
        GO TO 15
        STOP
        END
```

PROGRAM C

```
C  02   CALCULATION OF PERCENTAGE OF SEATS FILLED UP
        INTEGER DEPT,COURSE,DPTOLD,COROLD,SECTON,ENROLD,MAXIM
        DATA KSTOP/1H*/
        READ 10,DEPT,COURSE,SECTON , ENROLD,MAXIM
   10   FORMAT(A4,I2,I2,2I3)
   15   ISECT=1
        DPTOLD=DEPT
        COROLD=COURSE
        NCORS=1
   20   NSECT=1
        SETPCT=FLOAT(ENROLD)/FLOAT(MAXIM)
        READ 10,DEPT,COURSE,SECTON,ENROLD,MAXIM
        IF(COROLD.NE.COURSE) GO TO 25
        NSECT=NSECT+1
        ISECT=ISECT+1
        SETPCT=SETPCT+FLOAT(ENROLD)/FLOAT(MAXIM)
        GO TO 20
   25   IF(DPTOLD.NE.DEPT) GO TO 35
        NCORS=NCORS+1
        AVERAG=100.0*SETPCT/FLOAT(NSECT)
        PRINT 30,DPTOLD,COROLD,AVERAG
   30   FORMAT(1X,A4,2X,I2,3X,F8.2)
        COROLD=COURSE
   35   SAVERG=FLOAD(ISECT)/FLOAT(NCORS)
        PRINT 40,DPTOLD,SAVERG
   40   FORMAT(1X,A4,2X,F8.2)
        IF(DEPT.EQ.KSTOP) STOP
        GO TO 15
        STOP
        END
```

PROGRAM D

```
C  04  ANALYSIS OF STUDENT BIO DATA
       INTEGER RO,AE,CT,SE,DE,SX,MD,SG,SD(2),CD(2),ML(2),PV(20),
                                                          SM(3)
       DATA NM,SG,SD,CD,ML,PV,SM/31*0/
   10  READ 15,RO,DE,AE,SX,CT,SE,MD
   15  FORMAT(I3,I1,I2,2I1,I2,I1)
       IF(RO.EQ.0) GO TO 30
       IF(DE.EQ.0.OR.DE.GT.3) GO TO 50
       IF(SX.EQ.0.OR.SX.GT.2) GO TO 50
       IF(RO.GT.500)GO TO 50
       NM=NM+1
       SG=SG+AE
       SD(SX)=SD(SX)+1
       CD(CT)=CD(CT)+1
       ML(MD)=ML(MD)+1
       PV(SE)=PV(SE)+1
       SM(DE)=SM(DE)+1
       GO TO 10
   30  AG=FLOAT(SG)/FLOAT(NM)
       AM=100.0*FLOAT(SD(1))/FLOAT(NM)
       AF=100.0*FLOAT(SD(2))/FLOAT(NM)
       PRINT 35,NM,AM,AF,AG
   35  FORMAT(1X,I4,3(2X,F8.2))
       PRINT 40,(CD(I),ML(I),I=1,2)
   40  FORMAT(1X,2(I3/),2(I3/))
       PRINT 45,(PV(I),I=1,20),(SM(I),I=1,3)
   45  FORMAT(1X,20(I3/),3(I3/))
   50  STOP
       END
```

PROGRAM E

```
C  04  ANALYSIS OF STUDENT BIO-DATA
       INTEGER ROLNUM,AGE,CAST,STATE,DEGRE,SEX,MARIED,SUMAGE
      1 SEXCOD(2),CSTCOD(2),MARITL(2),PROVNC(20),SCHEM(3)
       DATA NUMBER,SUMAGE,SEXCOD,CSTCOD,MARITL,PROVNC,SCHEM/31*0/
   10  READ 15,ROLNUM,DEGRE,AGE,SEX,CAST,STATE,MARIED
   15  FORMAT(I3,I1,I2,2I1,I2,I1)
       IF(ROLNUM.EQ.0) GO TO 30
       IF(DEGRE.EQ.0.OR.DEGRE.GT.3) GO TO 50
       IF(SEX.EQ.0.OR.SEX.GT.2) GO TO 50
       IF(ROLNUM.GT.500) GO TO 50
       NUMBER=NUMBER+1
       SUMAGE=SUMAGE+AGE
       SEXCOD(SEX)=SEXCOD(SEX)+1
       CSTCOD(CAST)=CSTCOD(CAST)+1
       MARITL(MARIED)=MARITL(MARIED)+1
       PROVNC(STATE)=PROVNC(STATE)+1
       GO TO 10
   30  AVGAGE=FLOAT(SUMAGE)/FLOAT(NUMBER)
       AVGMAL=100.0*FLOAT(SEXCOD(1))/FLOAT(NUMBER)
       AVGFML=100.0*FLOAT(SEXCOD(2))/FLOAT(NUMBER)
       PRINT 35,NUMBER,AVGMAL,AVGFML,AVGAGE
   35  FORMAT(1X,I4,3(2X,F8.2))
       PRINT 40,(CSTCOD(I),MARITL(I),I=1,2)
   40  FORMAT(1X,2(I3/),2(I3/))
       PRINT 45,(PROVNC(I),I=1,20),(SCHEM(I),I=1,3)
   45  FORMAT(1X,20(I3/),3(I30))
   50  STOP
       END
```

APPENDIX F

CONTROL STRUCTURE AND EXECUTION STRUCTURE

QUIZ TA 306

ROLL NO. NAME:

The quiz is divided in PART A and PART B. You are given PART A first and you have to answer this part in 15 minutes. After the completion of PART A you will be given PART B to answer in 15 minutes.

PART A

Time: 15 Minutes
Marks: 5

Read carefully the program given below and try to understand it. The questions in PART B are also related to this program. The input data for the program is given at the end of the program. (This data is for PART A only). You are required to give the value of variable or print-out (if it is a WRITE statement) at different lines of the program. The space and the format for the answer is also given in the sheet.

(NOTE: The programs of the Quiz are on the following pages).

PROGRAM 11

PROGRAM: Calculation of gross-pay.

```
01       WRITE(6,15)
02   15  FORMAT(1X,*ID-NUMBER    GRØSS-PAY*)
03       TGP=0.0
04   20  READ(5,21) ID, REGHRS, ØVHRS, RATE
05   21  FORMAT(I5,3F5.0)
06       IF(ID.EQ.0) GO TO 28
07       GRSPAY=RATE*(REGHRS+1.5*ØVHRS)
08       WRITE(6,25) ID, GRSPAY
09   25  FORMAT(1X,I7,F15.2)
10       TGP=TGP+GRSPAY
11       GO TO 20
12   28  WRITE(6,30) TGP
13   30  FORMAT(3X, *TOTAL*, F15.2)
14       STOP
15       END
```

Input data

b̸1750b̸b̸40.b̸b̸b̸5.b̸b̸b̸7.

LINE NO	VARIABLE/OUTPUT	VALUE/PRINT-OUT
03	TGP	0.0 (Example)
01	OUTPUT	
06	ID	
07	GRSPAY	
08	OUTPUT	
10	TGP	

PROGRAM 12

PROGRAM: Computation of grade-point.

```
01          WRITE(6,1)
02     1    FORMAT(1X,*RØLLƀƀƀƀƀWEIGHTƀƀƀƀƀSPIƀƀƀƀƀCPI*)
03    10    READ(5,11) JRØLL,ØLCPI,CUMWT
04    11    FORMAT(I5,2F10.1)
05          IF(JRØLL.EQ.0) STOP
06          READ(5,15) WT1,GR1,WT2,GR2,WT3,GR3,WT4,GR4,WT5,GR5
07    15    FORMAT(10F5.1)
08          SEMWT=WT1+WT2+WT3+WT4+WT5
09          SPI=(WT1*GR1+WT2*GR2+WT3*GR3+WT4*GR4+WT5*GR5)/SEMWT
10          CPI=(ØLCPI*CUMWT+SPI*SEMWT)/(CUMWT+SEMWT)
11          WRITE(6,16)JRØLL,SEMWT,SPI,CPI
12    16    FORMAT(1X,I5,3F10.1)
13          GO TO 10
14          END
```

Input data

```
60432ƀƀƀƀƀƀƀ8.0ƀƀƀƀƀƀ48.0
ƀ12.0ƀƀ8.0ƀ12.0ƀƀ6.0ƀ12.0ƀ10.0ƀ12.0ƀƀ8.0ƀ12.0ƀƀ8.0
```

LINE NO	VARIABLE/OUTPUT	VALUE/PRINTOUT
03	CUMWT	48. (Example
03	JRØLL	
06	WT4	
08	SEMWT	
09	SPI	
10	CPI	

PROGRAM 13

PROGRAM: Calculation of compound interest.

```
01          READ(5,10)AMNT,IYEAR
02    10    FORMAT(F10.0,I2)
03          IF(AMNT.GT.10000.)GO TO 15
04          RATE=5.0
05          IMNTH=12
06          GO TO 20
07    15    RATE=8.0
08          IMNTH=6
09    20    N=IYEAR*12/IMNTH
10    30    IF(N.EQ.0) GO TO 40
11          ACUM=(RATE/100.)*AMNT*IMNTH
12          AMNT=AMNT+ACUM
13          N=N-1
14          GO TO 30
15    40    WRITE(6,45) AMNT,IYEAR
16    45    FORMAT(1X,*ACCUMALATED AMOUNT*,F10.2,*AFTER*,I2)
17          STOP
18          END
```

Input data

b̸b̸b̸b̸b̸8000.b̸5

LINE NO	VARIABLE/OUTPUT	VALUE/PRINTOUT
07	RATE	5.0 (Example)
03	AMNT	
08	IMNTH	
09	N	
15	ACUM	
15	AMNT	

PROGRAM 21

PROGRAM: Computation of discount.

```
01        READ(5,1)KTYPE,PRICE
02     1  FORMAT(I1,F10.2)
03        GO TO (10 , 20,30),KTYPE
04    10  IF(PRICE.GE.5000.0) GO TO 45
05        GO TO 40
06    20  IF(PRICE.GE.2000.0) GO TO 45
07        GO TO 40
08    30  IF(PRICE.GE.1000.0) GO TO 45
09    40  DISCT=0.0
10        GO TO 50
11    45  DISCT=0.1*PRICE
12    50  AMTDUE=PRICE-DISCT
13        WRITE(6,51)PRICE,KTYPE,DISCT,AMTDUE
14    51  FORMAT(1X,F10.2,I2,2F10.2)
15        STOP
16        END
```

Input data

2ƀƀƀ2000.00

LINE NO	VARIABLE/OUTPUT	VALUE/PRINTOUT
01	PRICE	2000.00 (Example)
03	KTYPE	
13	PRICE	
11	DISCT	
12	AMTDUE	
13	OUTPUT	

PROGRAM 22

PROGRAM: Computation of time in AM/PM from 'hours'

```
01         READ(5,1)JRLYTM
02     1   FORMAT(I4)
03         JHRS=JRLYTM/100
04         JMIN=JRLYTM-100*JHRS
05         IF(JHRS.GT.12) GO TO 20
06         IF(JHRS.EQ.0) JHRS=12
07         WRITE(6,10)JHRS,JMIN
08    10   FORMAT(1X,I2,*.*,I2,*ƀAM*)
09         STOP
10    20   JHRS=JHRS-12
11         IF(JHRS.EQ.0) JHRS=12
12         WRITE(6,21) JHRS,JMIN
13    21   FORMAT(1X,I2,*.*,I2,*ƀPM*)
14         STOP
15         END
```

Input data

2300

LINE NO	VARIABLE/OUTPUT	VALUE/PRINTOUT
01	JRLYTM	2300 (Example)
03	JHRS	
04	JMIN	
12	JHRS	
12	JMIN	
12	OUTPUT	

PROGRAM 23

PROGRAM: Calculation of sum of digits in a number.

```
01         READ(5,1)NUM,NDIGS
02      1  FORMAT(2I6)
03         KØUNT=0
04         JSUM=0
05         NDMIN1=NDIGS-1
06      5  N1=NUM/10
07         IF(NUM.EQ.0) GO TO 20
08         JDIGIT=NUM-N1*10
09         IF(JDIGIT.GT.0) JSUM=JSUM+JDIGIT
10         NUM=N1
11         KØUNT=KØUNT+1
12         IF(KØUNT.LT.NDMIN1) GO TO 5
13         JSUM=JSUM+NUM
14     20  WRITE(6,21)JSUM
15     21  FORMAT(1X,I6)
16         STOP
17         END
```

Input data

b96785b̸b̸b̸b̸b̸5

LINE NO.	VARIABLE/OUTPUT	VALUE/PRINTOUT
03	KØUNT	0 (Example)
05	NDMIN1	
06	N1 (in first iteration)	
08	JDIGIT (in first iteration)	
06	N1 (in second iteration)	
14	JSUM	

PROGRAM 31

PROGRAM: Computation of first and second marks of the class.

```
01          INTEGER MARKS(10)
02          DO 1 I=1,10
03     1    READ(5,2) MARKS(I)
04     2    FORMAT(I6)
05          MHIEST=0
06          DO 5 I=1,10
07          IF(MARKS(I).GT.MHIEST) MHIEST=MARKS(I)
08     5    CONTINUE
09          M2NDHI=0
10          DO 15 I=1,10
11          IF(MARKS(I).EQ.MHIEST) GO TO 15
12          IF(MARKS(I).GT.M2NDHI) M2NDHI=MARKS(I)
13    15    CONTINUE
14          WRITE(6,20) MHIEST,M2NDHI
15    20    FORMAT(1X,2I6)
16          STOP
17          END
```

Input data

```
ƀƀƀƀ30
    40
    20
    10
    18
    90
    80
    12
    85
    70
```

LINE NO.	VARIABLE/OUTPUT	VALUE/PRINTOUT
05	MHIEST	0 (Example)
09	MARKS(4)	
14	MARKS(8)	
14	MHIEST	
14	M2NDHI	
14	OUTPUT	

PROGRAM 32

PROGRAM: Conversion of binary number to decimal number

```
01        READ(5,10)JBIT0,JBIT1,JBIT2,JBIT3,JBIT4
02   10   FORMAT(5I1)
03        IDEC=0
04        IF(JBIT4.EQ.1) IDEC=IDEC+1
05        IF(JBIT3.EQ.1) IDEC=IDEC+2
06        IF(JBIT2.EQ.1) IDEC=IDEC+4
07        IF(JBIT1.EQ.1) IDEC=IDEC+8
08        IF(IDEC.GT.9) STOP
09        IF(JBIT0.EQ.1)GO TO 20
10        WRITE(6,15) IDEC
11   15   FORMAT(1X,*+*,I1)
12        STOP
13   20   WRITE(6,25) IDEC
14   25   FORMAT(1X,*-*,I1)
15        STOP
16        END
```

Input data
00101

LINE NO.	VARIABLE/OUTPUT	VALUE/PRINTOUT
03	IDEC	0 (Example)
04	IDEC	
05	IDEC	
06	IDEC	
07	IDEC	
10	OUTPUT	

PROGRAM 33

PROGRAM: Calculation of number of One's in a binary number (Parity check).

```
01        READ(5,10) JBIT0,JBIT1,JBIT2,JBIT3,JBIT4
02   10   FORMAT(5I1)
03        KØUNT=0
04        IF(JBIT1.EQ.1) KØUNT=KØUNT+1
05        IF(JBIT2.EQ.1) KØUNT=KØUNT+1
06        IF(JBIT3.EQ.1) KØUNT=KØUNT+1
07        IF(JBIT4.EQ.1) KØUNT=KØUNT+1
08        N=KØUNT/2
09        KØDD=KØUNT-2*N
10        JBITP=0
11        IF(KØDD.EQ.1) JBITP=1
12        IF(JBIT0.EQ.JBITP) STOP
13        WRITE(6,15)
14   15   FORMAT(1X,*PARITY ERROR*)
15        STOP
16        END
```

Input data

01010

LINE NO.	VARIABLE/OUTPUT	VALUE/PRINTOUT
03	KØUNT	0 (Example)
05	JBIT2	
06	JBIT4	
08	N	
09	KØDD	
12	JBITP	

QUIZ TA 303

ROLL NO.   NAME:

PART B

Time = 15 Minutes
Marks = 15

This part of the quiz is to test your understanding of the program. If you have understood the program in PART A, it should not be difficult for you to recall the same program. Try to recall the program from your memory and write it in the space given below. It is not necessary to remember exact variable names or statement numbers as long as the logic of the program remains the same.

APPENDIX G

FOUR-FACTOR EXPERIMENT

INSTRUCTIONS

This experiment is being conducted to study the factors affecting program comprehension. The experiment is divided into four sessions, each of two hours duration. In each session you will work on four independent programs spending half hour on each program. There will be a 15 minutes break after two programs. As each program is given to you, please read the program carefully and try to understand it. You should try to find out (i) what the program does and (ii) how it does. You will be given 15 minutes for this work. After 15 minutes, we shall take back the program from you. In next 10 minutes you have to reconstruct/recall the program which you have read. If you have understood the program it will not be difficult for you to reconstruct it. It is not necessary to use same variable names and statement numbers, which has appeared in the program, as long as logic of the program remains the same. At the end of the reconstruction task you write in English about the program. So in total of 30 minutes, you spend 15 minutes in program reading and understanding, 10 minutes in reconstruction and 5 minutes in commenting. This process will be repeated for other three quarters of the session with different programs.

If you find any error in the program please do point it out in your comments. To keep you fresh tea will be served in break.

APPENDIX H

PROGRAM FOR FOUR-FACTOR EXPERIMENT

PROGRAM 01

```
C  01  S=24 C=05 D=02  E=06 REF PWM 376
C      CALCULATION OF NUMBER OF WORDS IN SENTENCES IN A PARAGRAPH
       DIMENSION ICHAR(800),NCHAR(20)
       DATA NCHAR/20*0/
       DATA ISTOP,IBLNK/1H.,1H /
       READ 10,ICHAR
   10  FORMAT(80A1)
       I = 1
   15  IF(I.GT.800) GO TO 40
       IF(ICHAR(I).NE.IBLNK) GO TO 20
       I=I+1
       GO TO 15
   20  NOCHR*1
   25  I=I+1
       IF(I.GT.800) GO TO 30
       IF(ICHAR(I).EQ.IBLNK) GO TO 30
       IF(ICHAR(I).EQ.ISTOP) GO TO 30
       NOCHR=NOCHR+1
       GO TO 25
   30  NCHAR(NOCHR)=NCHAR(NOCHR)+1
       I=I+1
       GO TO 15
   40  PRINT 45,(I,NCHAR(I),I=1,20)
   45  FORMAT(1X,* S.NO    FREQUENCY*/20(1X,I3,6X,I3/))
       STOP
       END
```

PROGRAM 02

```
C  02  S=28 C=03 D=0 E=12 REF PWM 381
C      COMPUTATION OF AVERAGE NO OF SECTIONS PER COURSE IN A
C      DEPARTMENT AND AVERAGE PERCENTAGE OF SEAT VACANT IN EACH
                                                          COURSE
       INTEGER DEPT,COURSE,DEPTO,COURSO,SECT,REQ,MAX
       DATA KSTOP/1H*/
       READ 10,DEPT,COURSE,SECT,REQ,MAX
   10  FORMAT(A4,I2,I2,2I3)
   15  ISECT=1
       DEPTO=DEPT
       COURSO=COURSE
       NCORS=1
       NSECT=1
       SPCT=FLOAT(REQ)/FLOAT(MAX)
   20  READ 10,DEPT,COURSE,SECT,REQ,MAX
       IF(COURSO.NE.COURSE) GO TO 25
       NSECT=NSECT+1
       ISECT=ISECT+1
       SPCT=SPCT+FLOAT(REQ)/FLOAT(MAX)
       GO TO 20
   25  IF(DEPTO.NE.DEPT) GO TO 35
       NCORS=NCORS+1
       AVG=100.*SPCT/FLOAT(NSECT)
       PRINT 30,DEPTO,COURSO,AVG
   30  FORMAT(1X,A4,2X,I2,3X,F8.2)
       COURSO=COURSE
       GO TO 20
   35  SAVG=FLOAT(ISECT)/FLOAT(NCORS)
       PRINT 40,DEPTO,SAVG
   40  FORMAT(1X,A4,2X,F8.2)
       IF(DEPT.EQ.KSTOP) STOP
       GO TO 15
       END
```

PROGRAM 03

```
C  03   S=27 C=06 D=05 E=06 REF FWM 133
C       UPDATING THE STOCK-BOOK WITH NEW ARRIVAL AND SHIPMENTS
        DIMENSION ISTK(10),IQUAN(10),UNIT(10),PRICE(10),SAMNT(10)
        READ 10,(ISTK(I),IQUAN(I),UNIT(I),PRICE(I),I=1,10)
   10   FORMAT(2I3,2F8.2)
   15   READ 20 NSTK,NQUAN,UPRIC
   20   FORMAT(2I3,F8.2)
        IF(NSTK.EQ.0) GO TO 30
        DO 25 I=1,10
        IF(NSTK.NE.ISTK(I)) GO TO 25
        IQUAN(I)=IQUAN(I)+NQUAN
        PRICE(I)=PRICE(I)+UPRIC*FLOAT(NQUAN)
        UNIT(I)=PRICE(I)/FLOAT(IQUAN(I))
   25   CONTINUE
        GO TO 15
   30   READ 35,NSTK,NQUAN
   35   FORMAT(2I3)
        IF(NSTK.EQ.0) GO TO 45
        DO 40 I=1,10
        IF(NSTK.NE.ISTK(I)) GO TO 40
        SAMNT(I)=UNIT(I)*FLOAT(NQUAN)
        IQUAN(I)=IQUAN(I)-NQUAN
        PRICE(I)=PRICE(I)-UNIT(I)*FLOAT(NQUAN)
   40   CONTINUE
        GO TO 30
   45   PRINT 50,(ISTK(I),IQUAN(I),UNIT(I),PRICE(I),SAMNT(I),I=1,10
   50   FORMAT(10(1X,I3,2X,I3,3(1X,F8.2)))
        STOP
        END
```

PROGRAM 04

```
C   04   S=27 C=04 D=05 E=10 REF NONE
C        ANALYSIS OF STUDENTS BIO-DATA
         INTEGER ROLNO,AGE,CAST,STATE,DEGRE,SEX,MARD,SAGE,
        1SECOD(2),CSCOD(2),MARTL(2),PROV(20),SCHEM(3)
         DATA NUM,SAGE,SECOD,CSCOD,MARTL,PROV,SCHEM/31*0/
    10   READ 15,ROLNO,DEGRE,AGE,SEX,CAST,STATE,MARD
    15   FORMAT(I3,I1,I2,2I1,I2,I1)
         IF(ROLNO.EQ.0) GO TO 30
         IF(DEGRE.EQ.0.OR.DEGRE.GT.3) GO TO 50
         IF(SEX.EQ.0.OR.SEX.GT.2) GO TO 50
         IF(ROLNO.GT.999) GO TO 50
         NUM=NUM+1
         SAGE=SAGE+AGE
         SECOD(SEX)=SECOD(SEX)+1
         CSCOD(CAST)=CSCOD(CAST)+1
         MARTL(MARD)=MARTL(MARD)+1
         PROV(STATE)=PROV(STATE)+1
         SCHEM(DEGRE)=SCHEM(DEGRE)+1
         GO TO 10
    30   AVGAG=FLOAT(SAGE)/FLOAT(NUM)
         AVGM=100.*FLOAT(SECOD(1))/FLOAT(NUM)
         AVGF=100.*FLOAT(SECOD(2))/FLOAT(NUM)
         PRINT 35,NUM,AVGM,AVGF,AVGAG
    35   FORMAT(1X,I4,3(2X,F8.2))
         PRINT 40,NUM,(CSCOD(I),MARTL(I),I=1,2)
    40   FORMAT(1X,I4,2X,2(I3/))
         PRINT 45,NUM,(PROV(I),I=1,20),(SCHEM(I),I=1,3)
    45   FORMAT(1X,I4,2X,20(I3/),3(I3/))
    50   STOP
         END
```

PROGRAM 05

```
C   05  S=26 C=06 D=03 E=05 REF NONE
C       UPDATING OF LIB.STOCK
        DIMENSION DPTL(5)
        INTEGER DCOD,SCOD,DEPCOD(5),SUBCOD(5,10),ACCNO
        DATA GTOTAL,DPTL,DEPCOD,SUBCOD/6*0.0,55*0/
   10   READ 15,ACCNO,NOVOL,PRICE,DCOD,SCOD
   15   FORMAT(I4,I2,F8.2,2I2)
        IF(ACCNO.EQ.0) GO TO 30
        IF(ACCNO.GT.9999) GO TO 20
        IF(NOVOL.EQ.0.OR.NOVOL.GT.99) GO TO 20
        IF(PRICE.EQ.0.0.OR.PRICE.GT.2000.) GO TO 20
        TPRICE=PRICE*FLOAT(NOVOL)
        GTOTAL=GTOTAL+TPRICE
        IF(DCOD.EQ.0.OR.DCOD.GT.5) GO TO 20
        DPTL(DCOD)=DPTL(DCOD)+TPRICE
        DEPCOD(DCOD)=DEPCOD(DCOD)+NOVOL
        IF(SCOD.EQ.0.OR.SCOD.GT.10) GO TO 20
        SUBCOD(DCOD,SCOD)=SUBCOD(DCID,SCOD)+NOVOL
        GO TO 10
   20   PRINT 25,ACCNO
   25   FORMAT(1X,*ERROR IN DATA FOR ACCNO=*,I6)
        GO TO 10
   30   PRINT 35,GTOTAL
   35   FORMAT(1X,*GRAND TOTAL=*,F8.2)
        PRINT 40,(DEPCOD(I),I=1,5),((SUBCOD(I,J),J=1,10),I=1,5)
   40   FORMAT(1X,5(I5,2X,10(I5,2X)/))
        STOP
        END
```

```
C 06   S=26 C=07 D=01 E=12 REF PWM 378
C      COMPUTATION OF THE AVERAGE NO. OF WORDS IN THE SENTENCE
       DIMENSION IA(80)                                     IN PARA.
       DATA IBLNK,IPER/1H ,1H./
       NWDS=0
       NSEN=0
       DO 40 I=1,10
       READ 10,IA
   10  FORMAT(80A1)
       JW=0
       J=1
   15  IF(IA(J).NE.IPER) GO TO 20
       IF(JW.NE.0) NWDS=NWDS+1
       NSEN=NSEN+1
       GO TO 30
   20  IF(IA(J).EQ.IBLNK) GO TO 25
       JW=1
       GO TO 30
   25  IF(JW.EQ.1) NWDS=NWDS+1
       JW=0
   30  J=J+1
       IF(J.LT.80) GO TO 15
       IF(JW.EQ.1) NWDS=NWDS+1
   40  CONTINUE
       AVG=FLOAT(NWDS)/FLOAT(NSEN)
       PRINT 45,AVG
   45  FORMAT(1X,F8.2)
       STOP
       END
```

PROGRAM 07

```
C  07   S=25 C=07 D=05 E=07 REF PWM 195
C       THE PROGRAM ENCIPHERS(CODES) A MESSAGE.
        INTEGER KEY(8),MESAG(16),SYMBOL(27),NKEY(8),CRYP(16)
        DATA SYMBOL/1H ,1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,1HK,
       11HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT,1HU,1HV,1HW,1HX,1Hy,1HZ/
        READ 10,KEY,MESAG
   10   FORMAT(8A1/16A1)
        DO 20 J=1,8
        DO 15 K = 1,27
        IF(KEY(J).EQ.SYMBOL(K)) GO TO 20
   15   CONTINUE
        GO TO 100
   20   NKEY(J)=K-1
        J=0
        DO 35 L=1,16
        J=J+1
        IF(J.EQ.9) J=1
        DO 25 K=1,27
        IF(MESAG(L).EQ.SYMBOL(K)) GO TO 30
   25   CONTINUE
        GO TO 100
   30   M=K+NKEY(J)+9
        M=M-27*(M/27)
   35   CRYP(L)=SYMBOL(M+1)
        PRINT 40, CRYP
   40   FORMAT(1X,*CRYPTOGRAPH*,5X,16A1)
  100   STOP
        END
```

PROGRAM 08

```
C   08  S=27  C=07 D=05 E=06 REF NONE
C       PROGRAM COUNTS THE NO. OF SYMBOLS AND ITS FREQUENCY
        DIMENSION ICHAR(800),ISYMBL(27),IPUNC(3),IFREQ(27),IPCONT(3)
        DATA ISYMBL/1H ,1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,1HK,
       11HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT,1HU,1HV,1HW,1HX,1HY,1HZ/
        DATA IPUNC/1H,,1H.,1H-/,NUM,IFREQ,IPCONT/1,30*0/
        READ 10,ICHAR
   10   FORMAT(10(80A1/))
   15   IF(NUM.GT.800) GO TO 40
        ISW=0
        DO 20 I=1,27
        IF(ICHAR(NUM).NE.ISYMBL(I)) GO TO 20
        ISW=1
        IFREQ(I)=IFREQ(I)+1
   20   CONTINUE
        IF(ISW.EQ.1) GO TO 35
        DO 25 I=1,3
        IF(ICHAR(NUM).NE.IPUNC(I)) GO TO 25
        ISW=1
        IPCONT(I)=IPCONT(I)+1
   25   CONTINUE
        IF(ISW.NE.0) GO TO 35
        PRINT 30, ICHAR(NUM)
   30   FORMAT(1X,*INVALID CHAR*,2X,A1)
   35   NUM=NUM+1
        GO TO 15
   40   PRINT 45,(IFREQ(I),I=1,27),(IPCONT(I),I=1,3),NUM
   45   FORMAT(1X,27(1X,I3/),1X,3(1X,I3/),1X,I4)
        STOP
        END
```

PROGRAM 09

```
C   09   S=39 C=05 D=01 E=06 REF DVT 215
C        PROGRAM FOR COMPUTING WATER COMPANY BILL
         DIMENSION ANAME(5)
    10   READ 15,(ANAME(I),I=1,5),NCUST,MRL,MRN
    15   FORMAT(5A4,3I4)
         IF(NCUST.EQ.0) GO TO 90
         PRINT 20,(ANAME(I),I=1,5)
    20   FORMAT(1X,*NAME*,3X,5A4)
         IF(NCUST.GT.1000) GO TO 60
         PRINT 25,NCUST
    25   FORMAT(1X,*CUST.NO.*,2X,I5)
         IF(MRL.GT.9999) GO TO 70
         PRINT 30,MRL
    30   FORMAT(1X,*LAST METER READING*,2X,I5)
         IF(MRN.LT.MRL.OR.MRN.GT.9999) GO TO 80
         PRINT 35,MRN
    35   FORMAT(1X,*NEW METER READING*,2X,I5)
         IUSED=MRN-MRL
         PRINT 40,IUSED
    40   FORMAT(1X,*AMOUNT USED*,2X,I5)
         IF(IUSED.GT.100) GO TO 45
         AMNT1=0.1*FLOAT(IUSED)
         AMNT2=0.0
         GO TO 50
    45   AMNT1=0.1*100.0
         AMNT2=0.15*FLOAT(IUSED-100)
    50   AMNT=AMNT1+AMNT2
         PRINT 55,AMNT1,AMNT2,AMNT
    55   FORMAT(1X,*TOTAL AT RATE 1*,2X,F6.2/1X,*TOTAL AT RATE 2*,2X,
        1F6.2/1X,*TOTAL CHARGE*,7X,F7.2)
         GO TO 10
```

(continued)

PROGRAM 09 (continued)

```
60  PRINT 65,NCUST
65  FORMAT(1X,*ERROR IN CUSTOMER NO.*,2X,I5)
    GO TO 10
70  PRINT 75,MRL
75  FORMAT(1X,*ERROR IN LAST METER READING*,1X,I5)
    GO TO 10
80  PRINT 85,MRN
85  FORMAT(1X,*ERROR IN NEW METER READING*,1X,I5)
    GO TO 10
90  STOP
    END
```

PROGRAM 10

```
C   10  S=40 C=06 D=03 E=10 REF PWM 190
C       PROGRAM TO PLOT A GRAPH ON PRINTER
        DIMENSION X(20),Y(20),IP(50)
        DATA IA,IB/1H*,1H /
        READ 10,N
    10  FORMAT(I2)
        PRINT 15,N
    15  FORMAT(1X,*NO. OF POINTS*,2X,I3)
        READ 20,(X(I),Y(I),I=1,N)
    20  FORMAT(2F4.1)
        PRINT 25
    25  FORMAT(1H1)
        I=40
    30  DO 35 K=1,50
        IP(K)=IB
    35  CONTINUE
        DO 40 J=1,N
        IY=Y(J)/2.5+0.5
        IF(IY.NE.I) GO TO 40
        IX=X(J)*10.0+0.5
        IP(IX)=IA
    40  CONTINUE
        M=I/10*10
        IF(M.EQ.I) GO TO 50
        PRINT 45,IP
    45  FORMAT(11X,1H*,50A1)
        GO TO 60
    50  YP=2.5*FLOAT(I)
        PRINT 55,YP,IP
```

(continued)

PROGRAM 10 (continued)

```
55   FORMAT(1X,F6.2,2X,*+*,50A1)
60   I=I-1
     IF(I.GT.0) GO TO 30
     YP=0.0
     PRINT 65,YP
65   FORMAT(1X,F8.0,2X,*+*,5(10H*********+))
     DO 70 I=1,6
     X(I)=I-1
70   CONTINUE
     PRINT 75,(X(I),I=1,6)
75   FORMAT(2X,6F8.1)
     STOP
     END
```

PROGRAM 11

```
C  11  S=44 C=03 D=05 E=07 REF NONE
C      PREPARING LIST OF STUDENTS IN EACH DEPARTMENTS OF INST.
       DIMENSION DEPT(5),ANAME(5),NODPT(5),NOPRG(3),NFIN(3)
       DATA ATEND,NODPT,ISUM,M/1H*,7*0/
   10  READ 15,(DEPT(I),I=1,5)
   15  FORMAT(5A4)
       IF(DEPT(1).EQ.ATEND) GO TO 90
       M=M+1
       PRINT 20
   20  FORMAT(30X,*LIST OF STUDENTS*)
       PRINT 25,(DEPT(I),I=1,5)
   25  FORMAT(10X,5A4)
       DO 30 I=1,3
       NOPRG(I)=0
   30  NFIN(I)=0
   35  READ 40,JROL,(ANAME(I),I=1,5),IPCOD,IFCOD
   40  FORMAT(I6,1X,5A4,3I1)
       IF(JROL.EQ.0) GO TO 50
       NODPT(M)=NODPT(M)+1
       NOPRG(IPCOD)=NOPRG(IPCOD)+1
       NFIN(IFCOD)=NFIN(IFCOD)+1
       PRINT 45,JROL,(ANAME(I),I=1,5),IPCOD,IFCOD
   45  FORMAT(I7,3X,5A4,3X,I1,3X,I1)
       GO TO 35
   50  PRINT 55,NODPT(M)
   55  FORMAT(1X,*TOTAL NO. OF STUDENTS=*,I3)
       PRINT 60,NOPRG(1)
   60  FORMAT(1X,*NO. OF STUDENTS IN B.TECH.=*,I3)
       PRINT 65,NOPRG(2)
   65  FORMAT(1X,*NO.OF STUDENTS IN M.TECH.=*,I3)
       PRINT 70,NOPRG(3)
```

(continued)

FROGRAM 11 (continued)

```
 70  FORMAT(1X,*NO.OF STUDENTS IN PHD=*,I3)
     PRINT 75,NFIN(1)
 75  FORMAT(1X,*NO OF STUDENTS GETTING INST.SCHOLARSHIP=*,I4)
     PRINT 80,NFIN(2)
 80  FORMAT(1X,*NO. OF STUDENTS GETTING UGC SCHOLARSHIP=*,I4)
     PRINT 85,NFIN(3)
 85  FORMAT(1X,*NO. OF STUDENTS HAVING OTHER FIN.SUPPORT=*,I4)
     GO TO 10
 90  DO 95 I=1,5
 95  ISUM=ISUM+NODPT(I)
     PRINT 100
100  FORMAT(1X,*TOTAL NO.OF STUDENTS=*,I7)
     STOP
     END
```

PROGRAM 12

```
C   12  S=43 C=05 D=05 E=12 REF VR 169
C       QUESTIONNAIRE ANALYSIS
        INTEGER SEX(2),DIGRI(5),SUMAGE,AVAGE,SERNO,SCODE,AGE,CODIN,
       1DEGCOD,BADCAR
        DIMENSION INST(7),INTRT(5),INSINT(5,7)
        DATA INST,INTRT,INSINT,BADCAR,DIGRI,SUMAGE/54*0/
        SEX(1)=0
        SEX(2)=0
        NOQS=0
   10   READ 15,SERNO,SCODE,AGE,INSCOD,CODIN,DEGCOD
   15   FORMAT(I3,I1,I2,3I1)
        IF(SERNO.EQ.0) GO TO 30
        IF(SCODE.EQ.0.OR.SCODE.GT.2) GO TO 20
        IF(INSCOD.EQ.0.OR.INSCOD.GT.7) GO TO 20
        IF(CODIN.EQ.0.OR.CODIN.GT.5) GO TO 20
        IF(DEGCOD.EQ.0.OR.DEGCOD.GT.5) GO TO 20
        SEX(SCODE)=SEX(SCODE)+1
        INST(INSCOD)=INST(INSCOD)+1
        DIGRI(DEGCOD)=DIGRI(DEGCOD)+1
        INSINT(CODIN,INSCOD)=INSINT(CODIN,INSCOD)+1
        NOQS=NOQS+1
        SUMAGE=SUMAGE+AGE
        GO TO 10
   20   PRINT 25,SERNO
   25   FORMAT(1X,*ERROR IN CARD- NO.*,2X,I4)
        BADCAR=BADCAR+1
        GO TO 10
   30   PRINT 35,NOQS,BADCAR
```

(continued)

PROGRAM 12 (continued)

```
35  FORMAT(1X,10X,*NO.OF GOOD CARDS=*,I3/1X,*NO.OF BAD CARDS=*,I3)
    PRINT 40,SEX(1),SEX(2)
40  FORMAT(1X,*NO.OF MALE=*,I3/1X,*NO.OF FEMALE=*,I3)
    PRINT 45,(INST(I),I=1,7)
45  FORMAT(1X,*1*,3X,I3/1X,*2*,3X,I3/1X,*3*,3X,I3/1X,*4*,
   13X,I3/1X,*5*,3X,I3/1X,*6*,3X,I3/1X,*7*,3X,I3/)
    PRINT 50,(INTRT(I),I=1,5)
50  FORMAT(1X,*SCI*,3X,I3/1X,*ENG*,3X,I3/1X,*MATH*,3X,I3/
   11X,*SOC*,3X,I3/1X,*BUS*,3X,I3)
    PRINT 55,(DIGRI(I),I=1,5)
55  FORMAT(1X,*MATR*,3X,I3/1X,*ISC*,4X,I3/1X,*BSC*,4X,I3/
   11X,*MSC*,4X,I3/1X,*PHD*,4X,I3)
    AVAGE=SUMAGE/NOQS
    PRINT 60,AVAGE
60  FORMAT(1X,*AVERAGE AGE=*,F8.2)
    STOP
    END
```

PROGRAM 13

```
C  13  S=38 C=08 D=03 E=07 REF DVT 213
C      PROGRAM TO CHECK BANK ACCOUNT BALANCE
       DIMENSION BLNC(10),ICHARG(10)
       INTEGER ACNO(10),ANAME(10,5),ACNT
       DO 5 I=1,10
       READ 10,ACNO(I),(ANAME(I,J),J=1,5),BLNC(I)
   10  FORMAT(I2,5A4,F10.2)
    5  CONTINUE
       PRINT 15
   15  FORMAT(1H1,20X,*TRANSC.RECORD BEGIN*)
       DO 20 I=1,10
   20  ICHARG(I)=0
   25  READ 30,ACNT,TAMNT,ICOD
   30  FORMAT(I2,F8.2,I1)
       IF(ACNT.EQ.0) GO TO 75
       DO 65 I=1,10
       IF(ACNT.NE.ACNO(I)) GO TO 65
       PRINT 35,ACNO(I),BLNC(I)
   35  FORMAT(1X,I3,2X,F10.2)
       IF(ICOD.EQ.0) GO TO 50
       IF(BLNC(I).LT.TAMNT) GO TO 40
       BLNC(I)=BLNC(I)-TAMNT
       GO TO 55
   40  BLNC(I)=BLNC(I)-3.0
       PRINT 45,ACNO(I)
   45  FORMAT(1X,I3,10X,*CHEQUE NOT HONOURED BAD CHEQUE CHARGE DED.*)
       GO TO 70
   50  BLNC(I)=BLNC(I)+TAMNT
   55  ICHARG(I)=ICHARG(I)+1
```

(continued)

PROGRAM 13 (continued)

```
      IF(ICHARG(I).GT.10)ICHARG(I)=10
      PRINT60,ACNO(I),BLNC(I),TAMNT
   60 FORMAT(1X,I3,5X,F10.2,5X,F10.2)
   65 CONTINUE
   70 GO TO 25
   75 DO 80 I=1,10
   80 BLNC(I)=BLNC(I)-0.5*FLOAT(ICHARG(I))
      DO 100 I=1,10
      PRINT 85,ACNO(I),(ANAME(I,J),J=1,5),BLNC(I)
   85 FORMAT(I2,5A4,F10.2)
  100 CONTINUE
      PRINT 90
   90 FORMAT(1X,*TRANSACTION CLOSED*)
      STOP
      END
```

PROGRAM 14

```
C  14  S=41 C=10 D=02 E=12 REF PWM 379
C      PROGRAM PRINTS THE INTEGER NUMBER FROM A CARD
       DIMENSION IDIG(10),IA(80)
       DATA IBLNK,IPLUS,IMINUS,IDIG/1H ,1H+,1H-,1H0,1H1,
      11H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
       READ 10,IA
   10  FORMAT(80A1)
       PRINT 15,IA
   15  FORMAT(1X,80A1)
       IDG=0
       NUM=0
       J=0
   20  J=J+1
       IF(J.GT.80) GO TO 50
       IF(IA(J).EQ.IBLNK) GO TO 20
       ISIN=+1
       IF(IA(J).NE.IMINUS) GO TO 25
       ISIN=-1
       J=J+1
       GO TO 30
   25  IF(IA(J).EQ.IPLUS)J=J+1
   30  IF(J.GT.80) GO TO 60
       DO 35 I=1,10
       IF(IA(J).EQ.IDIG(I)) GO TO 75
   35  CONTINUE
       IF(IA(J).NE.IBLNK) GO TO 65
       IF(IDG.EQ.0) GO TO 65
   40  NUM=NUM*ISIN
       PRINT 45,NUM
```

(continued)

PROGRAM 14 (continued)

```
45   FORMAT(1X,*VALUE=*,2X,I10)
     GO TO 100
50   PRINT 55
55   FORMAT(1X,*BLANK CARD*)
     GO TO 100
60   IF(IDG.NE.0) GO TO 40
65   PRINT 70,IA,J
70   FORMAT(6X,80A1/6X,*INVALID CHARACTER NEAR COL.*,I3)
     GO TO 100
75   NUM=NUM*10+I-1
     IDG=1
     J=J+1
     GO TO 30
100  STOP
     END
```

PROGRAM 15

```
C  15  S=45 C=10 D=07 E=04 REF NONE
C      PRINTING LIST OF STUDENTS OF DEPARTMENT
       DIMENSION DEPT(5),ANAME(10,5),IPCOD(10),JROL(10),
      1IFCOD(10),SNAME(5)
       INTEGER ROL,PROG,FNC
       DO 130 I=1,5
       GO TO (10,20,30,40,50),I
   10  PRINT 15
   15  FORMAT(1H1,30X,*DEPT.OF AERONAUTICAL ENGG.*)
       GO TO 60
   20  PRINT 25
   25  FORMAT(1H1,30X,*DEPT.OF CHEMICAL ENGG.*)
       GO TO 60
   30  PRINT 35
   35  FORMAT(1H1,30X,*DEPT.OF CIVIL ENGG.*)
       GO TO 60
   40  PRINT 45
   45  FORMAT(1H1,30X,*DEPT.OF ELECTRICAL ENGG.*)
       GO TO 60
   50  PRINT 55
   55  FORMAT(1H1,30X,*DEPT.OF MECHANICAL ENGG.*)
   60  JK=0
       DO 75 J=1,10
       READ 65,ROL,(SNAME(I),I=1,5),PROG,FNC
   65  FORMAT(I6,1X,5A4,2I1)
       IF(ROL.EQ.0) GO TO 80
       JK=JK+1
       DO 70 K=1,5
```

(continued)

PROGRAM 15 (continued)

```
 70   ANAME(J,K)=SNAME(K)
      JROL(J)=ROL
      IPCOD(J)=PROG
 75   IFCOD(J)=FNC
 80   DO 90 M=1,JK
      IF(IPCOD(M).NE.1) GO TO 90
      PRINT 85,JROL(M),(ANAME(M,K),K=1,5),IPCOD(M),IFCOD(M)
 85   FORMAT(1X,I7,2X,5A4,3X,I2,3X,I2)
 90   CONTINUE
      DO 100 N=1,JK
      IF(IPCOD(N).NE.2) GO TO 100
      PRINT 95,JROL(N),(ANAME(N,K),K=1,5),IFCOD(N),IFCOD(N)
 95   FORMAT(1X,I7,2X,5A4,3X,I2,3X,I2)
100   CONTINUE
      DO 120 L=1,JK
      IF(IPCOD(L).NE.3) GO TO 120
      PRINT 110,JROL(L),(ANAME(L,K),K=1,5),IPCOD(L),IFCOD(L)
110   FORMAT(1X,I7,2X,5A4,3X,I2,3X,I2)
120   CONTINUE
130   CONTINUE
      STOP
      END
```

PROGRAM 16

```
C   16  S=52 C=13 D=10 E=14 REF VR 163
C       STUDENTS GRADE ANALYSIS
        DIMENSION ANAME(7),MARKS(6),ISUB(6),ISUAV(6),ICOV(6,6),
       1COV(6,6),ICROP(6,6),CORR(6,6),IDEV(6),SDEV(6)
        DATA NSTUD,ISUB,ICROP,ATLAST/43*0,1H*/
   10   READ 15,(ANAME(I),I=1,7),(MARKS(I),I=1,6)
   15   FORMAT(7A4,6I3)
        IF(ANAME(1).EQ.ATLAST) GO TO 60
        NSTUD=NSTUD+1
        MARTL=0
        DO 20 I=1,6
        MARTL=MARTL+MARKS(I)
   20   ISUB(I)=ISUB(I)+MARKS(I)
        AVG=FLOAT(MARTL)/6.0+0.5
        DO 25 I=1,6
        DO 25 J=1,6
   25   ICROP(I,J)=ICROP(I,J)+MARKS(I)*MARKS(J)
        IF(AVG.LT.60.0) GO TO 30
        KDIV=1
        GO TO 50
   30   IF(AVG.LT.50.) GO TO 35
        KDIV=2
        GO TO 50
   35   IF(AVG.LT.40.0) GO TO 40
        KDIV=3
        GO TO 50
   40   PRINT 45,(ANAME(I),I=1,7),AVG
   45   FORMAT(1X,7A4,2X,F5.2,2X,6HFAILED)
        GO TO 10
   50   PRINT 55,(ANAME(I),I=1,7),AVG,KDIV
   55   FORMAT(1X,7A4,2X,F5.2,2X,*PASSED IN *,I1)
```

(continued)

PROGRAM 16 (continued)

```
      GO TO 10
   60 DO 65 I=1,6
   65 ISUAV(I)=ISUB(I)/NSTUD
      PRINT 70,(ISUAV(I),I=1,6)
   70 FORMAT(1X,*PHYS*,3X,*CHEM*,3X,*MATH*,3X,*T.A.*,3X,*E.SC*,
     1*HUMS*/1X,6(I4,3X))
      DO 80 I=1,6
      DO 75 J=1,6
      ICOV(I,J)=ICROP(I,J)/NSTUD-ISUAV(I)*ISUAV(J)
   75 COV(I,J)=ICOV(I,J)
      SDEV(I)=SQRT(COV(I,I))
   80 IDEV(I)=SDEV(I)
      PRINT 85,(IDEV(I),I=1,6)
   85 FORMAT(1X,*STD.DEVIATION*,1X,6(I3,3X))
      DO 90 I=1,6
      DO 90 J=1,6
   90 CORR(I,J)=COV(I,J)/SQRT(COV(I,I)*COV(J,J))
      PRINT 95,((CORR(I,J),J=1,6),I=1,6)
   95 FORMAT(1X,*CORL.COEFF.MATRIX*/1X,*PHYS*,6F10.2/1X,*CHEM*,
     15F10.2/1X,*MATH*,4F10.2/1X,*T.A.*,3F10.2/1X,*E.SC*,2F10.2/
     11X,*HUMS*,F10.2)
      STOP
      END
```

APPENDIX I
COMPLEXITY RATING

INSTRUCTION

We are engaged in developing a measure for the complexity of programs. This complexity measure will reflect the difficulty in comprehension of the program. In this experiment you have to give your judgement about the program complexity from the point of view of understanding it. This program complexity may be due to size of the program, or its complex control structure or its complex data structure or executional behaviour or all of them. The experiment will be conducted in four sessions each of one and half hour duration. In each quarter of the session you will be given different program. Please read the program carefully and try to understand. After 15 minutes we shall take back the program from you. In next 5 minutes you prepare a brief summary of the program. The next task in the experiment is to give your judgement about the complexity of the program you read. You have to give your complexity rating on a scale of 10 points. If you judge the program extremly simple you should give 0 rating whereas if you think program extremly complex you should give a rating of 10. Please also give the percentage of contributing factors to the program complexity. So in total of 25 minutes, 15 minutes

you spend in reading and understanding the program, 5 minutes in preparing summary and 5 minutes in giving your judgement. Thus process will be repeated for other three quarters of the session with different programs.

If you find any error in the program please do point it out in the summary.

REMEMBER YOU ARE JUDGING PROGRAM COMPLEXITY.

APPENDIX J

To test the hypothesis of no difference between main effects of a factor (that is $\sigma_\alpha^2 = 0$) against the alternative hypothesis (that is $\sigma_\alpha^2 > 0$) one can use F ratios [Winer, 71, p. 335]. In terms of expected value of mean squares, the F ratio for this test has the general form

$$F_{ratio} = \frac{E(\text{numerator})}{E(\text{denominator})} = \frac{u + C^2 \sigma_\alpha^2}{u}$$

where u is some linear function of the variance of other parameters in the model and c is some coefficient. In other words, E (Mean square of numerator) must be equal to E (Mean square of denominator) when $\sigma_\alpha^2 = 0$. The mean square (MS) in the numerator of the F ratio must be MS of factors or their interactions. The mean square that is in the denominator depends upon the expected value of MS of factors or their interactions under the proper model. For linear additive model, the appropriate denominator for F ratio is MS error, for other model it may be MS interaction term. For details, reader may refer to [Winer 71, pp. 220-228, 332-335].

In the experiment No. (1) of Chapter 4, the factors of the experiment were subjects, program size, control structure, data structure and execution structure complexities. Each subject reconstructed all the programs. In this experiment, interaction of the subject with a source of variance was

used as error term in construction of F ratios. That is, for factor of control structure, (as example)

$$F_{ratio} = \frac{MS_{control\ structure}}{MS_{subject x control\ structure}}$$

Similarly, it had been calculated for other factors and their interactions. In all our experiments, we have explained the method of construction of F ratios.

APPENDIX K

On the basis of our experimental results we suggest the following do's and dont's for program development. Clearly, these are not complete set of guidelines but only those reached from our experiments.

Selection of Variable Names

The results of experiment number 1 and 2 of chapter 3 and 4 have established the importance of variable names in program understanding.

Variable names should be selected for their associations with well-understood problem-domain objects. If selected variable names get associated with two or more problem-domain objects, then they will create ambiguity. One must be especially watchful for ambiguous names in languages where the length of a name is seriously restricted, e.g. FORTRAN. To test about possible ambiguity in the names, one may take opinions of others about their association with problem domain object. It seems to be a good practice to include a set of comments explaining what problem-domain object is represented by each variable or group of variables.

Control Structure

It is advisable to organise control structures which are natural to the problem-domain. Each action of the problem domain should be implemented in separate control block. For

mple, if two disjoint computations are coded as a single loop, resulting program will be harder to understand than if they coded as two separate loops. (This is an indirect con-sion from experiment number 3 of chapter 3).

Data Structures

Number of variables (including arrays) are not a serious ference to understanding. If found desirable from the point view of clarity and naturalness, then additional variables need not be discouraged.

Program Size

Our experiment do not provide a strong basis to recommend limit on the size of the program or module. However, it clear from our experiments that if program size is small can understand it even if it has a bad control and data uctures. As the program size becomes order of 40 to 45 nes, the other factors of program complexity become more fective. Thus any program larger than 25 lines should be itten with full attention to understandability.

Execution Structure

It is evident from result of experiment number 1 of pter 3 that expression evaluation becomes increasingly diffi- as number of operands is increased. In light of this ult, we suggest that one should keep the number of operands each assignment statement to as low as possible.